# 原 高级编程技术--tc 图形界面

## 高级编程技术

介绍如何利用 TC 系统所提供的相关函数实现菜单设计、图形绘制、动画的播放、乐曲的演奏、汉字的显示、图片的显现等技术，在讲述时，以问题-解答的方式来逐渐阐明。


1 文本的屏幕输出和键盘输入..... 1

1.1 文本的屏幕输出.....1

1.2 键盘输入.....8

1.3 问题实现.....11

1.4 高级应用——菜单实现.....14

实验一.....23

2 图形显示方式和鼠标输入.....23

2.1 图形显示.....23

2.2 鼠标的使用.....52

2.3 问题实现.....58

实验二.....62

3 屏幕图象与动画技术.....63

3.1 一个简单的实现方法.....63

3.2 利用动态开辟图视口的方法.....66

3.3 利用屏幕图象存储再放的方法.....67

3.4 利用页交替的方法.....70

3.5 问题实现.....71

实验三.....74

4 中断技术.....74

4.1 编写自己的中断程序.....76

4.2 问题实现.....78

4.3 其它应用——硬中断演示秒表程序.....84

实验四.....87

5 发声技术.....87

5.1 声音函数.....87

5.2 计算机乐谱.....88

5.3 问题实现.....89

实验五.....93

6 汉字显示技术.....93

6.1 汉字编码.....93

6.2 问题实现.....95

实验六.....97


1

使用过 Windows 系统的用户都感受到了图形用户界面的直观和高效。所有 Windows 系统的应用程序都拥有相同或相似的基本外观，包括窗口、菜单、工具条、状态栏等。用户只要掌握其中一个，就不难学会其它软件，从而降低了学习成本和难度。而且 Windows 是一个多任务的操作环境，它允许用户同时运行多个应用程序，或在一个程序中同时做几件事情。例如，我们可以边欣赏 MP3 的音乐边 IE 冲浪，可以在运行 WORD 时同时编辑多个文档等。用户直接通过鼠标或键盘来使用应用程序，或在不同的应用程序之间进行切换，非常方便。这些都是单任务、命令行界面的 DOS 操作系统所无法比拟的。TC2.0 或 TC3.0 均是在 DOS 环境下运行的 C 系统。不过，无论采用 TC，还是 VC、BC，所产生的 C 可执行程序都是基于 DOS 系统的。

C 语言发展如此迅速，而且成为最受欢迎的语言之一，主要因为它具有强大的功能。C 是一种"中"级语言，它把高级语言的基本结构和语句与低级语言的实用性结合起来。C 语言可以对位、字节和地址进行操作，而这三者是计算机最基本的工作单元。C 语言具有各种各样的数据类型，并引入了指针概念，可使程序效率更高。另外 C 语言也具有强大的图形功能，支持多种显示器和驱动器。而且计算功能、逻辑判断功能也比较强大，可以实现决策目的。C 系统提供了大量的功能各异的标准库函数，减轻了编程的负担。所以要用 C 语言实现具有类 Windows 系统应用程序界面特征的、或更生动复杂的 DOS 系统的

程序，就必须掌握更高级的编程技术。这些技术与微机的硬件密切联系，除了在第一章介绍的内容外，更深入的知识将在接口和汇编这门后期课程中学习。

1 文本的屏幕输出和键盘输入

[问题的提出] 编制一个程序，将屏幕垂直平分成两个窗口，左边窗口为蓝色背景，白色前景，右边窗口为绿色背景，红色前景。两个窗口都设计为文本输入，即在窗口中可以输入文字，在窗口屏幕中显示出来。使用 tab 键在左右两个窗口中切换，每个窗口都有光标，活动窗口光标进行闪烁。

[分析] 在这个问题中我们遇到了在初学 C 时不曾接触到的新概念，如文本窗口、前景色、背景色，以及围绕它们要解决的新问题：

（1） 如何在屏幕中开文本输入的窗口？

（2） 如何设置窗口的前景色、背景色或闪烁等显示属性？

（3） 如何通过按键来控制窗口的切换？

[解答] 要解决这一编程问题，要求有两方面的学习过程：一是对于分析中的前两个问题，要求大家了解有关文本的屏幕输出的知识；二是对于第 3 个问题，要求对键盘的输入有所了解。下面先就这两个方面的内容做一介绍。

1.1 文本的屏幕输出

显示器的屏幕显示方式有两种：文本方式和图形方式。文本方式就是显示文本的模式，它的显示单位是字符而不是图形方式下的象素，因而在屏幕上显示字符的位置坐标就用行和列表示。Turbo C 的字符屏幕函数主要包括文本窗口大小的设定、窗口颜色的设置、窗口文本的清除和输入输出等函数。这些函数的有关信息(如宏定义等)均包含在 conio.h 头文件中，因此在用户程序中使用这些函数时，必须用 include 将 conio.h 包含进程序。

1) 文本窗口的定义

Turbo C 默认定义的文本窗口为整个屏幕，共有 80 列 25 行的文本单元。如图 3-1 所示，规定整个屏幕的左上角坐标为（1，1），右下角坐标为（80，25），并规定沿水平方向为 X 轴，方向朝右；沿垂直方向为 Y 轴，方向朝下。每个单元包括一个字符和一个属性，字符即 ASCII 码字符，属性规定该字符的颜色和强度。除了这种默认的 80 列 25 行的文本显示方式外，还可由用户通过函数：void textmode(int newmode)；来显式地设置 Turbo C 支持的 5 种文本显示方式。该函数将清除屏幕，以整个屏幕为当前窗口，并移光标到屏幕左上角。newmode 参数的取值见表 3-1，既可以用表中指出的方式代码，又可以用符号常量。LASTMODE 方式指上一次设置的文本显示方式，它常用于在图形方式到文本方式的切换。

(1 列，1 行)

(80 列，25 行)

第一行

第一列

第 80 列

第 25 行

图 3-1 屏幕文本显示坐标

表 3-1 文本显示方式

方式 符号常量 显示列×行数和颜色

0 BW40 40×25 黑白显示

1 C40 40×25 彩色显示

2 BW80 80×25 黑白显示

3 C80 80×25 彩色显示

7 MONO 80×25 单色显示

-1 LASTMODE 上一次的显示方式

Turbo C 也可以让用户根据自己的需要重新设定显示窗口，也就是说，通过使用窗口设

置函数 window()定义屏幕上的一个矩形域作为窗口。window()函数的函数原型为：

void window(int left, int top, int right, int bottom)；

函数中形式参数（int left，int top）是窗口左上角的坐标，（int right，int bottom）是窗口的

右下角坐标，其中（left，top）和（right，bottom）是相对于整个屏幕而言的。例如，要定

义一个窗口左上角在屏幕（20，5）处，大小为 30 列 15 行的窗口可写成：

window(20, 5, 50, 25);

若 window()函数中的坐标超过了屏幕坐标的界限，则窗口的定义就失去了意义，也就是说

定义将不起作用，但程序编译链接时并不出错。

窗口定义之后，用有关窗口的输入输出函数就可以只在此窗口内进行操作而不超出窗口

的边界。

另外，一个屏幕可以定义多个窗口，但现行窗口只能有一个（因为 DOS 为单任务操作

系统）。当需要用另一窗口时，可将定义该窗口的 window()函数再调用一次，此时该窗口便

成为现行窗口了。

2) 文本窗口颜色和其它属性的设置

文本窗口颜色的设置包括背景颜色的设置和字符颜色（既前景色）的设置，使用的函数及其原型为：

3

设置背景颜色函数：void textbackground(int color)；

设置字符颜色函数：void textcolor(int color)；

有关颜色的定义见表 3-2。表中的符号常数与相应的数值等价，二者可以互换。例如设定兰色背景可以使用 textbackground(1)，也可以使用 textbackground(BLUE)，两者没有任何区别，只不过后者比较容易记忆，一看就知道是兰色。

表 3-2 颜色表

符号常数 数值 含义 背景或背景

BLACK 0 黑 前景、背景色

BLUE 1 蓝 前景、背景色

GREEN 2 绿 前景、背景色

CYAN 3 青 前景、背景色

RED 4 红 前景、背景色

MAGENTA 5 洋红 前景、背景色

BROWN 6 棕 前景、背景色

LIGHTGRAY 7 淡灰 前景、背景色

DARKGRAY 8 深灰 用于前景色.

LIGHTBLUE 9 淡蓝 用于前景色

LIGHTGREEN 10 淡绿 用于前景色

LIGHTCYAN 11 淡青 用于前景色

LIGHTRED 12 淡红 用于前景色

LIGHTMAGENTA 13 淡洋红 用于前景色

YELLOW 14 黄 用于前景色

WHITE 15 白 用于前景色

BLINK 128 闪烁 用于前景色

Turbo C 另外还提供了一个函数，可以同时设置文本的字符和背景颜色，这个函数是

文本属性设置函数：void textattr(int attr)；

参数 attr 的值表示颜色形式编码的信息，每一位代表的含义如下：

字节低四位 cccc 设置字符颜色，4~6 三位 bbb 设置背景颜色，第 7 位 B 设置字符是否闪烁。

假如要设置一个兰底黄字，定义方法如下：

```
textattr(YELLOW+(BLUE<<4));
```

若再要求字符闪烁，定义变为：

```
textattr(128+YELLOW+(BLUE<<4);
```

注意：

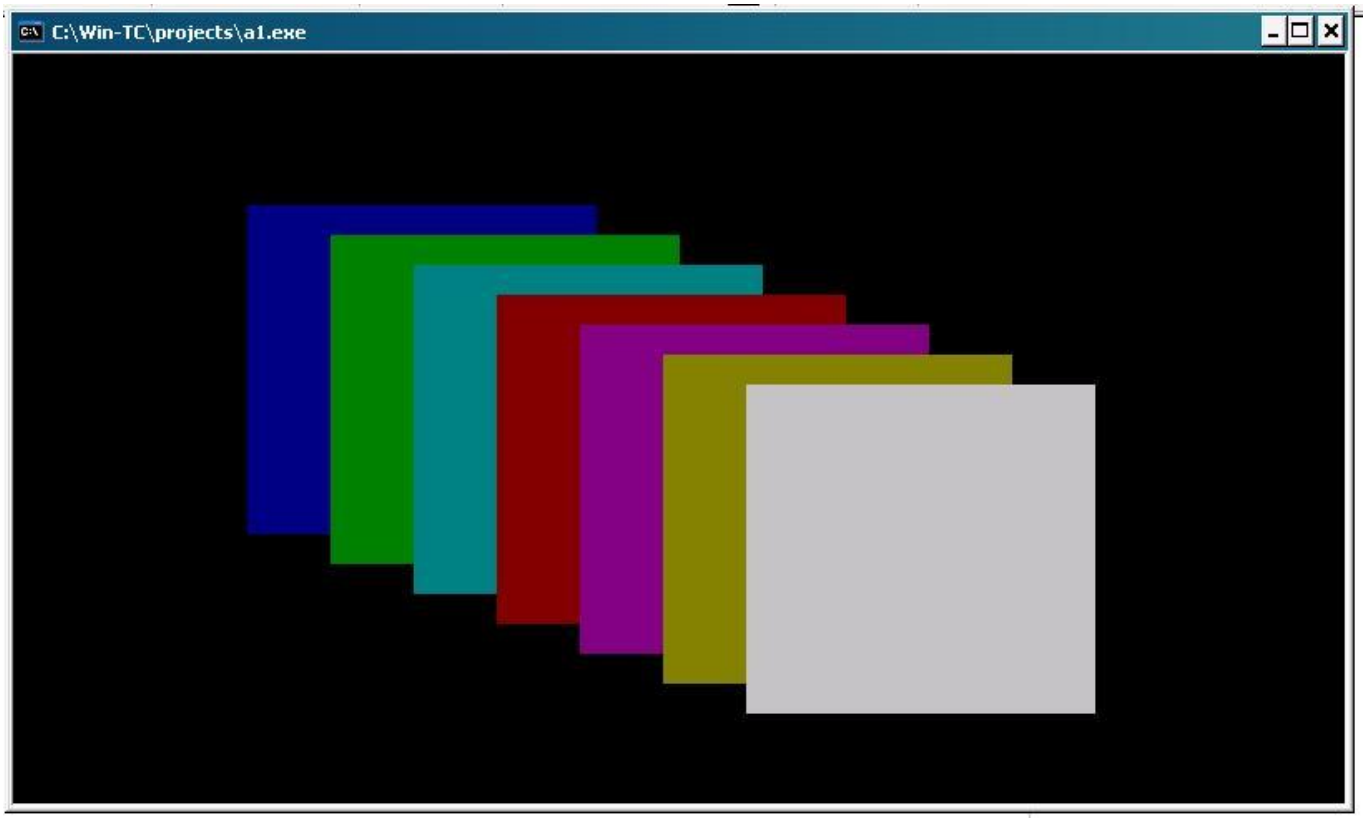
(1) 对于背景只有 0 到 7 共八种颜色，取大于 7 小于 15 的数，则代表的颜色与减 7 后的值对应的颜色相同；

(2) 用 textbackground()和 textcolor()函数设置了窗口的背景与字符颜色后，在没有用 clrscr()函数清除窗口之前，颜色不会改变，直到使用了函数 clrscr()，整个窗口和随后输出到窗口中的文本字符才会变成新颜色。

4

(3) 用 textattr()函数时背景颜色应左移 4 位，才能使 3 位背景颜色移到正确位置；

例程 3-1：这个程序使用了关于窗口大小的定义、颜色的设置等函数，在一个屏幕上不同位置定义了 7 个窗口，其背景色分别使用了 7 种不同的颜色。




```
/*-------例程 3-1-------*/
#include <stdio.h>
#include <conio.h>
int main()
{
int i;
textbackground(0); /* 设置屏幕背景色，待 clrscr 后起作用 */
clrscr(); /* 清除文本屏幕 */
for(i=1; i<8; i++)
{
window(10+i*5, 5+i, 30+i*5, 15+i); /* 定义文本窗口 */
textbackground(i); /* 定义窗口背景色 */
clrscr(); /* 清除窗口 */
}
getch();
return 0;
}
```

void highvideo(void)；

该函数将设置用高亮度显示字符。

void lowvideo(void)；

该函数将设置用低亮度显示字符。

void normvideo(void)；

该函数将设置通常亮度显示字符。

3) 窗口内文本的输入输出函数

? 窗口内文本的输出函数

我们以前介绍过的 printf()，putc()，puts()，putchar()和输出函数以整个屏幕为窗口的，它们不受由 window 设置的窗口限制，也无法用函数控制它们输出的位置，但 Turbo C 提供了三个文本输出函数，它们受窗口的控制，窗口内显示光标的位置，就是它开始输出的位置。

当输出行右边超过窗口右边界时，自动移到窗口内的下一行开始输出，当输出到窗口底部边界时，窗口内的内容将自动产生上卷，直到完全输出完为止，这三个函数均受当前光标的控制，每输出一个字符光标后移一个字符位置。这三个输出函数原型为：

int cprintf(char *format，表达式表)；

int cputs(char *str)；

int putch(int ch)；

它们的使用格式同 printf()，puts()和 putc()，其中 cprintf()是将按格式化串定义的字符串或数据输出到定义的窗口中，其输出格式串同 printf 函数，不过它的输出受当前光标控制，且输出特点如上所述，cputs 同 puts，是在定义的窗口中输出一个字符串，而 putch()则是输出一个字符到窗口，它实际上是函数 putc 的一个宏定义，即将输出定向到屏幕。

? 窗口内文本的输入函数

5

可直接使用 stdio.h 中的 getch 或 getche 函数。需要说明的是，getche()函数从键盘上获得一个字，在屏幕上显示的时候，如果字符超过了窗口右边界，则会被自动转移到下一行的开始位置。

4) 有关屏幕操作的函数

void clrscr(void)；

该函数将清除窗口中的文本，并将光标移到当前窗口的左上角，即(1, 1)处。

void clreol(void)；

该函数将清除当前窗口中从光标位置开始到本行结尾的所有字符，但不改变光标原来的位置。

void delline(void)；

该函数将删除一行字符，该行是光标所在行。

void gotoxy(int x, int y)；

该函数很有用，用来定位光标在当前窗口中的位置。这里 x，y 是指光标要定位处的坐标（相对于窗口而言）。当 x, y 超出了窗口的大小时，该函数就不起作用了。

int movetext(int x1, int y1, int x2, int y2, int x3, int y3)；

该函数将把屏幕上左上角为(xl，y1)，右下角为(x2，y2)的矩形内文本拷贝到左上角为(x3，y3)的一个新矩形区内。这里 x，y 坐标是以整个屏幕为窗口坐标系，即屏幕左上角为(1，1)。该函数与开设的窗口无关，且原矩形区文本不变。

int gettext(int xl, int yl, int x2, int y2, void *buffer)；

该函数将把左上角为(xl，y1)，右下角为(x2，y2)的屏幕矩形区内的文本存到由指针 buffer 指向的一个内存缓冲区内，当操作成功，返回 1；否则，返回 0。

因一个在屏幕上显示的字符需占显示存储器 VRAM 的两个字节，即第一个字节是该字符的 ASCII 码，第二个字节为属性字节，即表示其显示的前景、背景色及是否闪烁，所以 buffer 指向的内存缓冲区的字节总数的计算为：

字节总数=矩形内行数×每行列数×2

其中：矩形内行数=y2-y1+l，每行列数=x2-xl+1（每行列数是指矩形内每行的列数）。矩形内文本字符在缓冲区内存放的次序是从左到右，从上到下，每个字符占连续两个字节并依次存放。

int puttext(int x1, int y1, int x2, int y2, void *buffer)；

该函数则是将 gettext()函数存入内存 buffer 中的文字内容拷贝到屏幕上指定的位置。

注意：

(1) gettext()函数和 puttext()函数中的坐标是对整个屏幕而言的，即是屏幕的绝对坐标，而不是相对窗口的坐标；

(2) movetext()函数是拷贝而不是移动窗口区域内容，即使用该函数后，原位置区域的文本内容仍然存在。

例程 3-2：下面的程序首先定义了一个字符数组，下标为 64，表示用来存四行八列的文本。

由于没有用 window 函数设置窗口，因而用缺省值，即全屏幕为一个窗口，程序开始设置 80 列×25 行文本显示方式(C80)，背景色为蓝色，前景色为红色，经 clrscr 函数清屏后，设置的背景色才使屏幕背景变蓝。gotoxy(10，10)使光标移到第 10 行 10 列，然后在(10，10)开始位置显示 L：load，接着在下面三行相同的列位置显示另外三条信息，13 行 10 列显示的 E：exit 后面带有回车换行符，为的是将光标移到下一行开始处，好显示 press any key to continue。当按任一键后，gettext 函数将(10，l0，18，13)矩形区的内容存到 ch 缓存区内。ch 即上述的四行八列信息，接着设置一个窗口，并纵向写上 1，2，3，4，然后用 movetext()，

6

将此窗口内容复制到另一区域，由于此区域包括背景色和显示的字符，所以被复制到另一区域的内容也是相同的背景色和文本。当按任一键后，又出现提示信息，再按键，则存在 ch 缓冲区内的文本由 puttext()又复制到开设的窗口内了，注意上述的函数 movetext()，gettext()，puttext()均与开设的窗口内坐标无关，而是以整个屏幕为参考系的。

```
/*-------例程 3-2-------*/
#include <conio.h>
main()
{
int i;
char ch[4*8*2]; /* 定义 ch 字符串数组作为缓存区 */
textmode(C80);
textbackground(BLUE);
textcolor(RED);
clrscr();
gotoxy(10,10);
cprintf("L:load");
gotoxy(10,11);
cprintf("S:save");
gotoxy(10,12);
cprintf("D:delete");
gotoxy(10,13);
cprintf("E:exit\r\n");
cprintf("Press any key to continue");
getch();
gettext(10,10,18,13,ch); /* 存矩形区文存到 ch 缓存区 */
clrscr();
textbackground(1);
textcolor(3);
window(20,9,34,14); /* 开一个窗口 */
clrscr();
cprintf("1.\r\n2.\r\n3.\r\n4.\r\n");/* 纵向写 1，2，3，4 */
movetext(20,9,34,14,40,10); /* 将矩形区文本复制到另一区域 */
puts("hit any key");
getch();
clrscr();
cprintf("press any key to put text");
getch();
clrscr();
puttext(23,10,31,13,ch); /* 将 ch 缓存区所存文本在屏上显示 */
getch();
```

}

5) 状态查询函数

有时需要知道当前屏幕的显示方式，当前窗口的坐标、当前光标的位置，文本的显示

7

属性等，Turbo C 提供了一些函数得到屏幕文本显示有关信息的函数：

void gettextinfo(struct text_info *f)；

这里的 text_info 是在 conio.h 头文件中定义的一个结构，该结构的定义是

structtext_info(

unsigned char winleft； /* 窗口左上角 x 坐标 */

unsigned char wintop； /* 窗口左上角 y 坐标 */

unsigned char winright; /* 窗口右下角 x 坐标 */

unsigned char winbottom; /* 窗口右下角 y 坐标 */

unsigned char attributes； /* 文本属性 */

unsigned char normattr; /* 通常属性 */

unsigned char currmode； /* 当前文本方式 */

unsigned char screenheight； /* 屏高 */

unsigned char screenwidth； /* 屏宽 */

unsigned char curx； /* 当前光标的 x 值 */

unsigned char cury； /* 当前光标的 y 值 */

};

例程 3-3：下面的程序将屏幕设置成 80 列彩色文本方式，并开了一个 window(1，5，70，20)的窗口，在窗口中显示了 current information of window，然后用 gettextinfo 函数得到当前窗口的信息，后面的 cprintf()函数将分别显示出结构 text_info 各分量的数值来，即：

current information of window

Left corner of window is l,5 Right corner of window is 70，20 Text window attribute is 29 Text window normal attribute is 29 Current video mode is 3 window height and width is 25，80 Row cursor pos is 2，Column pos is 1

```
/*-------例程 3-3-------*/
#include <conio.h>
main()
{
struct text_info current;
textmode(C80);
textbackground(1);
textcolor(13);
window(1,5,70,20);
clrscr();
cputs("Current information of window\r\n");
gettextinfo(&current);
cprintf("Left corner of window is %d,%d",
current.winleft,current.wintop);
cprintf("Right corner of window is %d,%d",
current.winright,current.winbottom);
cprintf("Text window attribute is%d",
current.attribute);
cprintf("Text window normal attribute",
current.normattr);
```

8

```
cprintf("Current video mode is%d",current.currmode);
cprintf("Window height and width is%d,%d",
current.screenheight,current.screenwidth);
cprintf("Row cursor pos is %d,Column pos is %d",
current.cury,current.curx);
getch();
}
```

1.2 键盘输入

当我们按下键盘上某键时，系统如何知道某键被按下呢？它的奥妙在于计算机键盘是一个智能化的键盘，在键盘内有一个微处理器，它用来扫描和检测每个键的按下和拾起状态。然后以程序中断的方式（INT 9）与主机通信。ROM 中 BIOS 内的键盘中断处理程序，会将一个字节的按键扫描码（扫描码的 0~6 位标识了每个键在键盘上的位置，最高位标识按键的状态，0 对应该键是被按下；1 对应松开。它并不能区别大小写字母，而且一些特殊键如 PrintScreen 等不产生扫描码直接引起中断调用）翻译成对应的 ASCII 码。

由于 ASCII 码仅有 256 个(28)，它不能将 PC 键盘上的键全部包括，因此有些控制键如 CTRL，ALT，END，HOME，DEL 等用扩充的 ASCII 码表示，扩充码用两个字节的数表示。第一个字节是 0，第二个字节是 0~255 的数，键盘中断处理程序将把转换后的扩充码存放在 Ax 寄存器中，存放格式如表 3-3 所示。对字符键，其扩充码就是其 ASCII 码。

表 3-3 键盘扫描码

| 扫描码 键 | 主键盘区: | 0x0c2d – | 0x1769 i | 0x256b k | 0x332c ,. | 0x4800 上箭头 | 0x4c35 5 |
|---|---|---|---|---|---|---|---|
| 0x011b ESC | 0x2960 ~ | 0x0d3d = | 0x186f o | 0x266c l | 0x342e | 0x4b00 左箭头 | 0x4d36 6 |
| 0x3b00 F1 | 0x0231 1 | 0x2b5c \ | 0x1970 p | 0x273b ; | 0x352f / | 0x5000 下箭头 | 0x4e2b |
| 0x3c00 F2 | 0x0332 2 | 0x0e08 退格键 | 0x1a5b [ | 0x2827 ' | 0x3920 空格键 | 0x4d00 右箭头 | 0x4f31 1 |
| 0x3d00 F3 | 0x0433 3 | 0x0f09 Tab | 0x1b5d ] | 0x1c0d 回车 | 右边数字键盘: | 0x352f / | 0x5032 2 |
| 0x3e00 F4 | 0x0534 4 | 0x1071 q | 0x1e61 a | 0x2c7a z | 0x5200 Insert | 0x372a * | 0x5133 3 |
| 0x3f00 F5 | 0x0635 5 | 0x1177 w | 0x1f73 s | 0x2d78 x | 0x4700 Home | 0x4a2d - | 0x5230 0 |
| 0x4000 F6 | 0x0736 6 | 0x1265 e | 0x2064 d | 0x2e63 c | 0x4900 Page UP | 数字键盘 | 0x532e Del |
| 0x4100 F7 | 0x0837 7 | 0x1372 r | 0x2166 f | 0x2f76 v | 0x5300 Delete | 0x4737 7 | |
| 0x4200 F8 | 0x0938 8 | 0x1474 t | 0x2267 g | 0x3062 b | 0x4f00 End | 0x4838 8 | |
| 0x4300 F9 | 0x0a39 9 | 0x1579 y | 0x2368 h | 0x316e n | 0x5100 PageDown | 0x4939 9 | |
| 0x4400 F10 | 0x0b30 0 | 0x1675 u | 0x246a j | 0x326d m | | 0x4b34 4 | |

键名 AH AL

字符键 扩充码=ASCII 码 ASCII 码

功能键/组合键 扩充码 0

是否有键按下，何键按下，简单的应用中可采用两种办法：一是直接使用 Turbo C 提供的键盘操作函数 bioskey()来识别，二是通过第一章 1.2.4.3 节介绍的 int86()函数，调用 BIOS 的 INT 16H，功能号为 0 的中断。它将按键的扫描码存放在 Ax 寄存器的高字节中。

函数 bioskey()的原型为：

int bioskey(int cmd)；

它在 bios.h 头文件中进行了说明，参数 cmd 用来确定 bioskey()如何操作：

cmd 操作

0 bioskey()返回按健的键值，该值是 2 个字节的整型数。若没有键按下，则该函数一直等待，直到有键按下。当按下时，若返回值的低 8 位为非零，则表示为普通键，其值代表该键的 ASCII 码。若返回值的低 8 位为 0，则高 8 位表示为扩展的 ASCII 码，表示按下的是特殊功能键。

1 bioskey()查询是否有键按下。若返回非 0 值，则表示有键按下，若为 0 表示没键按下。

2 bioskey()将返回一些控制键是否被按过，按过的状态由该函数返回的低 8 位的各位

值来表示：

字节位 对应的 16 进制数 含义

0 0x01 右边的 shift 键被按下

1 0x02 左边的 shift 键被按下

2 0x04 Ctrl 键被按下

9

3 0x08 Alt 键被按下

4 0x10 Scroll Lock 已打开

5 0x20 Num Lock 已打开

6 0x40 Caps Lock 已打开

7 0x80 Inset 已打开

当某位为 l 时，表示相应的键已按，或相应的控制功能已有效，如选参数 cmd 为 2，若 key 值为 0x09，则表示右边的 shift 键被按，同时又按了 Alt 键。

函数 bioskey()的原型为：

int int86(int intr_num，union REGS *inregs，union REGS *outregs)；

这个函数在 bios.h 头文件中进行了说明，它的第一个参数 intr_num 表示 BIOS 调用类型号，相当于 int n 调用的中断类型号 n，第二个参数表示是指向联合类型 REGS 的指针，它用于接收调用的功能号及其它一些指定的入口参数，以便传给相应的寄存器，第三个参数也是一个指向联合类型 REGS 的指针，它用于接收功能调用后的返回值，即出口参数，如调用的结果，状态信息，这些值从相关寄存器中得到。

例程 3-4：第二章扫雷游戏中，我们定义上,下,左,右键用来移动雷区光标的位置，回车或者空格键用来挖开光标当前指向的雷区方块，F 和 f 标记当前光标指向的方块有地雷，Q 和 q 在光标指向方块打问号，表示可能有地雷，A 和 a 用来自动挖开光标周围的方块，ESC 退出游戏。在实现时，我们调用 bioskey(0)来获得按键值，然后经过判断转入相应的处理。下面让我们再来回顾一下 2.2.3.4 节中的扫雷游戏源程序片段，其中 key.c 文件仅有一个函数 getKey()，它用 biosky(0)读取键盘输入，读到一个有用键（上,下,左,右键、回车或者空格键、F、f、Q、q、A、a、ESC）时返回该键值。

```
/*key.c——扫雷游戏的按键获取*/
#include <bios.h>
/*define key-value*/
#define ENTER 0x1c0d
#define UP 0x4800
#define DOWN 0x5000
#define LEFT 0x4b00
#define RIGHT 0x4d00
#define ESC 0x011b
#define SPACE 0x3920
#define LOWERF 0x2166
#define UPPERF 0x2146
#define LOWERA 0x1e61
#define UPPERA 0x1e41
#define LOWERQ 0x1071
#define UPPERQ 0x1051
int getKey(void){
while(1){
int key=bioskey(0);
switch(key){
case ENTER:
```

```
case UP:
case DOWN:
case LEFT:
case RIGHT:
case ESC:
case SPACE:
case LOWERF:
case UPPERF:
case LOWERA:
case UPPERA:
case LOWERQ:
case UPPERQ: return key;
}
}
}
/*-----------------------------------------End of key.c-----------------------------------------*/
```

例程 3-5：本程序仅仅演示了通过 int86()获取按键的扫描码。在此注意扫描码和 bioskey()返回的码值是不同的。

```
#include <stdio.h>
#include <dos.h>
/* define Keys scan code */ /* 定义各键的扫描码 */
#define Key_ESC 1
#define Key_A 30
int getKeySCode();
main()
{
int acount=0, ky;
do
{
ky= int getKeySCode();; /* 得到按键的扫描码 */
switch(ky){
case Key_A: /* A and a key */
++account; break;
case Key_ESC:
printf("\nEnd the program");
exit(0);
11
default:
break;
}
printf("\nDuring the program, you press A and a %d times",acount);
}
/* read char on key,return scan code */
int getKeySCode() /* 读键函数 */
{
union REGS rg;
rg.h.ah=0;
int86(0x16,&rg,&rg);
```

return rg.h.ah;

}

1.3 问题实现

在了解了上两小节的内容后，你就可以解决我们最初提出的问题了（见例程 3-6）。

我们的设计思想是首先用文本窗口函数 window (int x1, int y1, int x2, int y2)画出两个窗口，用 textcolor (int color)，textbackground (int color)，clrscr (void)等进行窗口属性的设置。用 tab 键进行两个窗口间的循环切换，在每次切换前先调用 gettext (int left, int top, int right, int bottom, void * buf)函数把当前矩形窗口上的字符拷贝到由 buf 所指向的内存中，在切换到另一个窗口后调用 puttext (int left, int top, int right, int bottom, void * buf)把先前存储在该窗口 buf 所指向的内存中的字符拷贝到当前窗口中，并用 gotoxy (int x, int y)把光标移到原先所在位置，因此可以接着先前的文本继续编辑。

程序中运用一个临时变量 turn 的值 0 和 1 来进行两个窗口间的循环切换，当值为 1 时调用 draw_left_win()函数重绘左边窗口，当值为 0 时调用 draw_right_win()函数重绘右边窗口。

程序用 bioskey(0)来读取键盘输入的文本并存在内存中，然后用 putch(key)输出。程序运行（已再次切换到左窗口，光标在问号后闪烁）的界面如图 3-2 所示。

在这个程序中我们调用了 process.h 中的 void exit(int status)函数，它清除和关闭所有打开的文件，写出任何缓冲输出，并终止程序。参数为 0，则认为程序正常终止；若为非零值，则说明存在执行错误。

/*例程 3-6*/

图 3-2 3.1 节问题实现后程序运行界面

12

```
#include <stdio.h>
#include <conio.h>
#include <bios.h>
/*切换时保存左窗口文本*/
char leftbuf[40*25*2];
/*切换时保存右窗口文本*/
char rightbuf[40*25*2];
/*切换时保存左窗口当前坐标*/
int leftx, lefty;
/*切换时保存右窗口当前坐标*/
int rightx, righty;
/*重绘左边窗口*/
void draw_left_win();
/*重绘右边窗口*/
void draw_right_win();
int main()
{
int key;
int turn;
textmode(C80);
textbackground(0);
textcolor(WHITE);
clrscr();
gotoxy(60,1);
cprintf("Press Esc to Quit");
window(41,2,79,24); /*右边窗口为绿色背景，红色前景*/
textbackground(2);
```

```
textcolor(4);
clrscr();
gettext(41,2,79,24, rightbuf);
window(2,2,40,24); /*左边窗口为蓝色背景，白色前景*/
textbackground(1);
textcolor(15);
clrscr();
gettext(2,2,40,24, leftbuf);
turn = 0; /*初始激活左窗口*/
for(;;)
{
key=bioskey(0);
13
if(key == 0x11b)
exit(0);
key=key&0xff; /*获取窗口输入的文本的 ASCII 码值*/
if(key == '\t')
{
if(turn == 0) /*切换到左窗口*/
{
gettext(2,2,40,24, leftbuf);
leftx = wherex();
lefty = wherey();
draw_right_win();
turn = 1;
}
else if(turn == 1) /*切换到右窗口*/
{
gettext(41,2,79,24, rightbuf);
rightx = wherex();
righty = wherey();
draw_left_win();
turn = 0;
}
}
else
putch(key); /*当前光标处显示新输入的文本字符*/
}
}
void draw_right_win()
{
window(41,2,79,24);
textbackground(2);
textcolor(4);
clrscr();
puttext(41,2,79,24, rightbuf);
gotoxy(rightx, righty);
}
```

```
void draw_left_win()
{
window(2,2,40,24);
textbackground(1);
textcolor(15);
clrscr();
puttext(2,2,40,24, leftbuf);
14
gotoxy(leftx, lefty);
}
```

1.4 高级应用——菜单实现

1.4.1 一个弹出式菜单

这个程序（例程 3-7）是一个文本方式下的菜单程序，它生成一个弹出式菜单，如图 3-3 所示。程序运行时，首先弹出一个带洋红色框的蓝底菜单，并有一红色光条压在第一项上，当按 E 键时，程序回到系统，压 Down 键时，光条移到第二项，压 A 时，则列出当前目录下的各文件目录，再压任意键后则又弹出菜单，当压 B 键时，则列出目录，当屏满后，暂停，按键后，又继续列出，它实际上就是执行 dir /p 命令，当按回车键后，又出现菜单，当按 C 键后，便执行 dir /w 的 dos 命令，以宽行格式列出目录，按回车键后又回到菜单用 UP 键可使光条上移。

程序开始时，首先用 do 循环反复运行下面的程序，直到按 E 键后为止，window(7，8，19，15)定义了一个用洋红色填充的窗口，接着又定义一个蓝色窗口，它套在洋红色窗口内，因而形成一个带洋红色粗边框的蓝色窗口，在窗口内写上各菜单项，接着又调用光条上移函数 upbar(y—1)，使红色光条压在第一菜单项上，下面的 do 循环则用来构成一个反复检查按键，并转去执行相应的功能操作，do 循环内的第一个 switch(ky)语句用来判断按的是何键，按键值赋给 y 的相应值，并使 ky=key_enter，以结束 do 循环，跟着第二个 switch(y)语句则按 y 值转去执行相应的菜单项功能。只有当菜单项功能键是 E 时，才使得外层的 do 循环结束而回到系统。

upbar(int y)函数是产生一个上移光条(用 gettext()函数)，实际上它第一次被调用时，则用 gettext(i，y，i，y，&t)连续 8 次将菜单项 8 个字符长的区域(蓝底无字)存入&t 中，用蓝底白字又放回(用 puttext()函数)，而第 2 个 gettext()函数则将下移一行的 8 个字符长区域(即第一个菜单项，存入 t 中，又以白字红底方式放回原处，即产生一个红色光条压在第一菜单项上。以后的调用(当按 UP 键时)则随 y 值不同，而红色光条压在不同菜单项上。

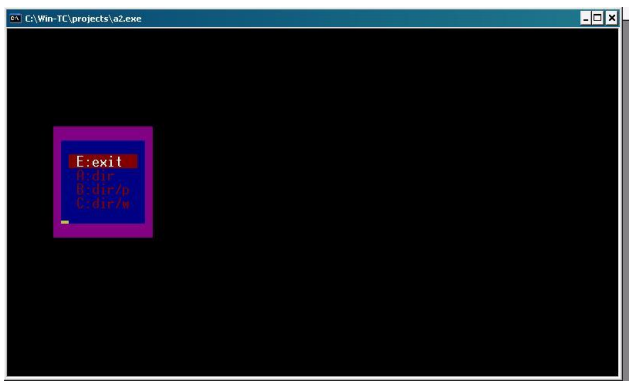
downbar()函数产生下移光条，当按 Down 键时，便调用该函数将红底白字的光条压在下一条菜单项上。

在 Turbo C 的 stdlib.h 中提供了一个能够执行 DOS 命令的函数 system ()，其原型为：

int system(char *command)；

参数 command 所指向的字符串正是 DOS 的命令。在这个程序中我们通过调用 system("dir")，system("dir/p")，system("dir/w")使程序根据菜单选项 A，B，C 来分别先执行 DOS 命令 dir

图 3-3 弹出式菜单



（列出当前目录下的各文件目录），dir/p（列出目录，当屏满后，暂停，按键后，又继续列出），dir/w（以宽行格式列出目录）后再返回程序。

```
/*例程 3-7*/
#include <process.h>
#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <conio.h>
/* define Key Values /* 定义各键的 bioskey(0)的按键返回值 */
#define Key_DOWN 0x5100
#define Key_UP 0x4900
#define Key_A 0x1e41
#define Key_a 0x1e61
#define Key_B 0x3042
#define Key_b 0x3062
#define Key_C 0x2e43
#define Key_c 0x2e63
#define Key_E 0x1245
#define Key_e 0x1265
#define Key_ENTER 0x1c0d
main()
{
int ky,y;
char ch;
textbackground(0);
clrscr();
do
{
textmode(C80);
textbackground(13);
textcolor(RED);
window(7,8,19,15); /* 开一个窗口 */
clrscr();
textbackground(1);
textcolor(RED);
window(8,9,18,14); /* 再开一个当前窗口,套在上一个窗口之中*/
clrscr();
gotoxy(3,3);
cprintf("E:exit\r\n"); /* 窗口中写上红色的菜单项 */
gotoxy(3,4);
cprintf("A:dir\r\n");
gotoxy(3,5);
16
cprintf("B:dir/p\r\n");
gotoxy(3,6);
cprintf("C:dir/w\r\n");
y=10;
upbar(y-1); /* 调用光条上移函数 */
```

```
do
{
ky=bioskey(0); /* 得到按键的键值 */
switch(ky)
{
case Key_A: case Key_a:/* A and a key */
{
y=12;
ky=Key_ENTER;
break;
}
case Key_B:case Key_b:/* B and b key */
{
y=13;
ky=Key_ENTER;
break;
}
case Key_C: case Key_c:/* C and c key */
{
y=14;
ky=Key_ENTER;
break;
}
case Key_E:case Key_e:/* E and e key */
{
y=11;
ky=Key_ENTER;
break;
}
case Key_DOWN:/* Cursor down key */
{
if ( y<13 )
{
upbar(y);
y++;
}
break;
}
case Key_UP: /* Cursor up key */
17
{
if(y>10)
{
downbar(y);
y--;
}
break;
}
```

```
}
} while (ky !=Key_ENTER ); /* Enter key */
textcolor(WHITE);
switch(y)
{
case 11:
{
ch='%'; /* 返回系统 */
break;
}
case 12:
{
system("dir");
getch(); /* 列目录后等待按键 */
break;
}
case 13:
{
system("dir /p");
getch(); /* 列目录屏满后暂停,按任意键继续 */
break;
}
case 14:
{
system("dir /w");
getch(); /* 用宽行格式列目录 */
break;
}
}
if(ch=='%')
break;
} while(1);
clrscr(); /* 清屏 */
}
18
/* read bar down */
upbar(int y) /* 光条上移函数 */
{
int i;
typedef struct texel_struct {
unsigned char ch;
unsigned char attr;
} texel;
texel t;
for(i=9;i<=17; i++)
{
gettext(i,y,i,y,&t);
t.attr=0x1f; /* 字符为白色,背景为蓝色 */
```

```
puttext(i,y,i,y,&t);
gettext(i,y+1,i,y+1,&t);
t.attr=0x4f; /* 字符为白色,背景为红色 */
puttext(i,y+1,i,y+1,&t);
}
gotoxy(3,y+1);
return;
}
/* red bar up */ /* 光条下移函数 */
downbar(int y)
{
int i;
typedef struct texel_struct {
unsigned char ch;
unsigned char attr;
} texel;
texel t;
for(i=9;i<=17;i++)
{
gettext(i,y,i,y,&t);
t.attr=0x1f; /* 字符为白色,背景为蓝色 */
puttext(i,y,i,y,&t);
gettext(i,y-1,i,y-1,&t);
t.attr=0x4f ; /* 字符为白色,背景为红色 */
puttext(i,y-1,i,y-1,&t);
}
gotoxy(3,y-1);
return;
}
```

19

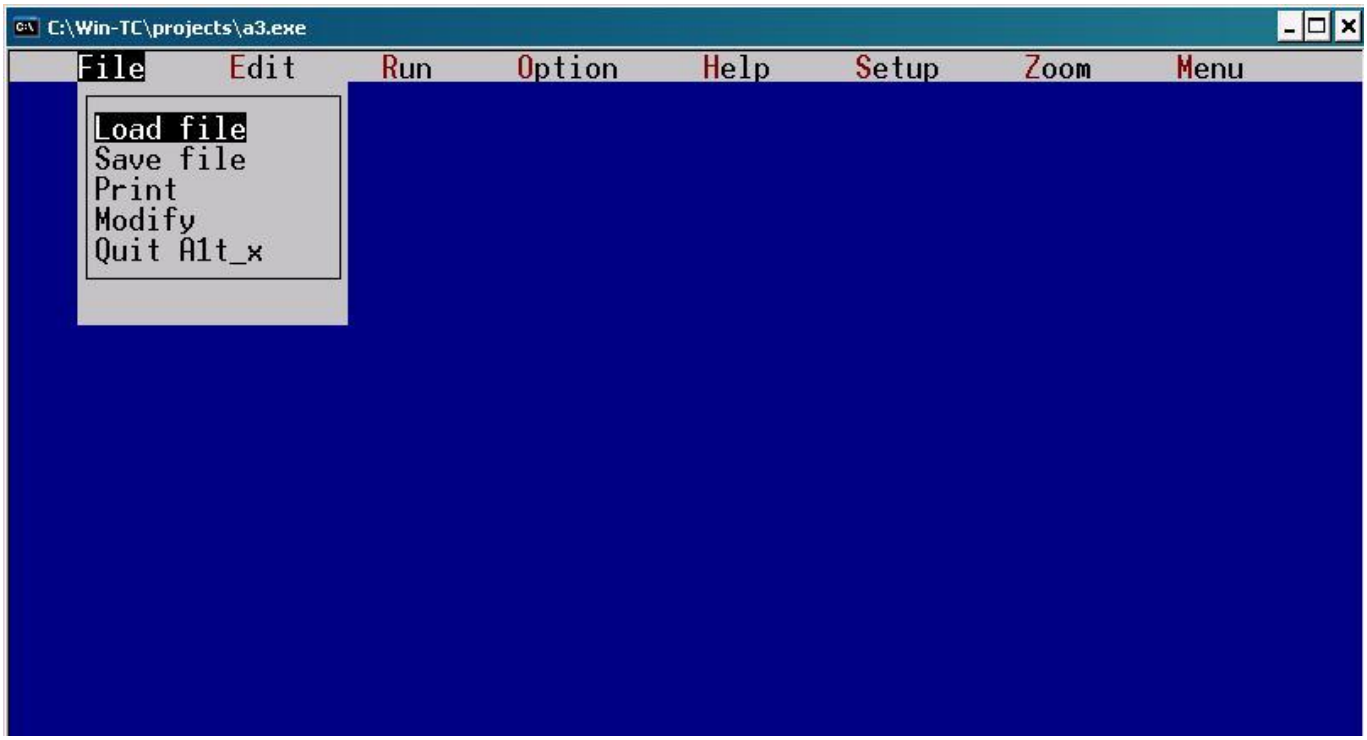
1.4.2 一个下拉式菜单

该程序（例程 3-8）在文本方式下产生一个下拉式菜单，程序运行时首先在屏幕顶行产生一个浅灰底黑字的主菜单，各菜单项的第一个字母加红，表示为热键，如图 3-4 所示，当选择主菜单第一项，即按 ALT_F 时，便产生一个下拉式子菜单，可用 UP 和 DOWN 键使压在第一个子菜单项上的黑色光条上下移动，当压在某子菜单项上，且按回车后，程序便转去执行相应子菜单项的内容，由于篇幅关系，该程序仅是一个演示程序，只作了第一个主菜单项和对应的子菜单，且子菜单项对应的操作只在程序相应处作了说明，并无具体内容，对主菜单项其它各项未作选它时相应的子菜单，但作法和第一项 File 的相同，故不赘述。

程序中用指针数组 munu[]存放主菜单各项，red[]存放各项的热键字符(即主菜单项各项的第一个字母)，f[]存放主菜单第一项 file 的子菜单各项。定义字符数组 buf 存放原子菜单所占区域的内容，buf1 存放一个子菜单项区域内容，由于一个字符占两个字节，故所占列数均乘了 2。外层循环处理主菜单，第一步显示主菜单界面，即先使整个屏幕的背景色为蓝色，然后开辟显示主菜单项的窗口(window(1，1，80，1))，用浅灰底黑字依次显示出主菜单各项，用红色字母再重现各项的第一个字母，并使光标定位在主菜单项的第一项 File 的 F 处；第二步用键盘管理函数 bioskey()获取菜单选项，当按 ALT_X 键时，则退出本程序，若按 ALT_F 键时，则执行弹出子菜单的操作：首先加黑主菜单的 File 项显示，将子菜单的区域内容保存到 buf 缓冲区内(用 gettext(5,2,20,12,buf))，这样当子菜单项消失时，用它来恢复原区域的内容。然后设置一个作子菜单的浅灰色窗口(window(5,2,20,9))，调用作框函数 box，

在浅灰色底上面输出一个矩形框来，接着的 for 循环则在框内显示出子菜单各项内容。接着是处理 File 的子菜单项的内层循环：首先获取按键，当为 ALT_X 时退出本程序；当为 ESC 键时直接返回到外层循环，即返回到主界面；当为 UP 或 DOWN 键时，则产生黑色光条的上下移动，当光条在第一项上时，若再按 UP 键，则光条移到最后一项，若光条原来就在最后一项，再按 DOWN 键，则光条退回到第一子菜单项去，这由 y=y==2?6：y-1 和 y=y==6?2：y+l 来实现，当光条压在某子菜单项上，且当按键为 ENTER 时程序则转去执行相应的子菜单项指明的操作，它们由 switch(y-1)语句来实现，当光条压在第一子菜单项上，且按回车后，则执行 case l 后的操作，由于是示范程序，具体操作没有指出，要变为实用菜单，则需在此处填上操作内容，转去作相应的处理，处理之后返回到外层循环。

用画框子程序 box()画框，实际上是由符号 ┌(ASCII 为 0xda)，—， └，｜，┘，┐，拼成了一个矩形框，仍然是文本输出，这是和图形方式下画框不同之处。




```
/*例程 3-8*/
#include <process.h>
#include <dos.h>
#include <conio.h>
#define Key_DOWN 0x5100
#define Key_UP 0x4900
#define Key_ESC 0x011b
#define Key_ALT_F 0x2100
#define Key_ALT_X 0x2d00
#define Key_ENTER 0x1c0d
void box(int startx,int starty,int high,int width);
main()
{
int i,key,x,y,l;
char *menu[] = {"File","Edit","Run","Option","Help","Setup","Zoom","Menu"};
/* 主菜单各项 */
char *red[] = { "F","E","R","O","H","S","Z","M" }; /* 加上红色热键 */
char *f[] = {"Load file", "Save file", "Print", "Modify ", "Quit A1t_x"};
/* File 项的子菜单 */
```

```
char buf[16*10*2],buf1[16*2]; /* 定义保存文本的缓冲区 */
while(1)
{
textbackground(BLUE);
clrscr();
textmode(C80);
window(1,1,80,1);/* 定义显示主菜单的窗口 */
textbackground(LIGHTGRAY);
textcolor(BLACK);
clrscr();
gotoxy(5,1);
for(i=0,l=0;i<8;i++)
{
x=wherex(); /* 得到当前光标的坐标 */
y=wherey();
cprintf("%s",menu[i]); /* 显示各菜单项 */
l=strlen(menu[i]); /* 得到菜单项的长度 */
gotoxy(x,y);
textcolor(RED);
cprintf("%s",red[i]); /* 在主菜单项各头字符写上红字符 */
x=x+l+5;
gotoxy(x,y);
textcolor(BLACK); /* 为显示下一个菜单项移动光标 */
}
gotoxy(5,1);
21
key=bioskey(0);
switch (key){
case Key_ALT_X:
exit(0); /* ALT_X 则退出 */
case Key_ALT_F:
{
textbackground(BLACK);
textcolor(WHITE);
gotoxy(5,1);
cprintf("%s",menu[0]); /* 加黑 File 项 */
gettext(5,2,20,12,buf); /* 保存窗口原来的文本 */
window(5,2,20,9);/* 设置作矩形框的窗口 */
textbackground(LIGHTGRAY);
textcolor(BLACK);
clrscr();
box(1,1,7,16); /* 调用作框函数 */
for(i=2;i<7;i++) /* 显示子菜单各项 */
{ gotoxy(2,i);
cprintf("%s",f[i-2]);
}
gettext(2,2,18,3,buf1); /*将下拉菜单的内容保存在 buf1*/
textbackground(BLACK);
```

```
textcolor(WHITE);
gotoxy(2,2);
cprintf("%s",f[0]);/*加黑下拉菜单的第一项 load file*/
gotoxy(2,2);
y=2;
while ((key=bioskey(0))!=Key_ALT_X) /* 等待选择下拉菜单项*/
{
if ((key==Key_UP)||(key==Key_DOWN))
{
puttext(2,y,18,y+1,buf1); /* 恢复原先的项 */
if (key==Key_UP)
y=y==2?6:y-1;
else
y=y==6?2:y+1;
gettext(2,y,18,y+1,buf1);/*保存要压上光条的子菜单项*/
textbackground(BLACK);
textcolor(WHITE);
gotoxy(2,y);
cprintf("%s",f[y-2]); /* 产生黑条压在所选项上 */
gotoxy(2,y);
22
}
else
if (key==Key_ENTER)/* 若是回车键,判断是哪一子菜单按的回
车，在此没有相应的特殊处理*/
{
switch ( y-1 ){
case 1: /* 是子菜单项第一项:Load file */
break;
case 2: /* Save file */
break;
case 3: /* print */
break;
case 4: /* modify */
break;
case 5:
exit(0);
default:
break;
}
break;
}
else
if (key==Key_ESC)
break; /* 是 Esc 键,返回主菜单 */
}
if (key==Key_ALT_X) exit(0);
break;
```

```
}
}
}
}
void box(int startx,int starty,int high,int width) /* 画矩形框函数 */
{ int i;
gotoxy(startx,starty);
putch(0xda); /* 画 ┌ */
for (i=startx+1;i<width;i++) putch(0xc4); /* 画— */
putch(0xbf); /* 画┐ */
for( i=starty+1;i<high;i++)
{
gotoxy(startx,i);putch(0xb3); /* 画 | */
gotoxy(width,i);putch(0xb3); /* 画 | */
}
gotoxy(startx,high);
23
putch(0xc0); /* 画 └ */
for (i=startx+1;i<width;i++) putch(0xc4); /* 画— */
putch(0xd9); /* 画┘ */
return ;
}
```

实验一

(1) 编写一个类似 tt.exe 中的游戏，计算机控制一些单词从屏幕上边下落，用户输入准确的单词时，由屏幕下方发出一个箭头，击毁该单词。

(2) 针对第 2 章提出的扫雷游戏问题，参考 Windows 系统自带的扫雷游戏，采用本节介绍的知识，实现图 3-5 所示的主菜单和相应的下拉菜单，并且尝试完成部分菜单功能。

2 图形显示方式和鼠标输入

[问题的提出] 编写程序，使用鼠标进行如下操作：按住鼠标器的任意键并移动，十字光标将随鼠标而移动，根据按键的不同采用不同的形状来画出相应的移动轨迹：当仅按下左键时，用圆圈；仅按下右键时，用矩形；其它按键情况用线条。

[分析] 在这个问题中我们看到输入的操作已不再是通过键盘，而是用鼠标。而且我们还要响应鼠标的具体操作，在屏幕上画出点、矩形、圆等图形。

[解答] 要解决这一编程问题，将涉及两方面的内容：一是关于程序设计中较难且又最吸引人的部分——计算机图形程序设计，即图形方式（另外一种显示器显示方式）的知识；二是关于鼠标的知识。下面将对它们做具体的解释。

2.1 图形显示

图形方式和文本方式不同，我们可以在这种方式下画图，它的显示单位是象素。如同近看电视的画面一样，显示器显示的图形也是由一些圆点组成(其亮度、颜色不同)，这些点称为象素(或称象点)。满屏显示象素多少，则决定了显示的分辨率高低，可以看出象素越小(或个数越多)，则显示的分辨率越高。象素在屏幕上的位置则可由其所在的 x，y 坐标来决定。

图 3-5 Windows 扫雷游戏的菜单

图 3-6 (a)显示屏在 640×480 分辨率下的坐标 (b)不同位置象素的坐标

24

显示屏的图形坐标系统就象一个倒置的直角坐标系（如图 3-6 所示）：定义屏幕的左上角为原点，正 x 轴右延伸，正 y 轴向下延伸，即 x 和 y 坐标值均为非负整数，但其最大值则由显示器的类型和显示方式来确定，也就是说，显示的象素大小可以通过设置不同的显示方式来改变。例如在图 3-6(a)所示的显示方式下，x，y 最大坐标是(639，399)，即满屏显示的象素

个数为 640×400。3-6(b)示出了不同位置象素的坐标，其最大的 x，y 值(即行和列值)由程序设置的显示方式来决定。我们称这种显示坐标为屏幕显示的物理坐标或绝对坐标，以便和图视窗口(图视口)坐标相区别。图视窗口是指在物理坐标区间又开辟一个或多个区间，在这些区间又可定义一个相对坐标系统，以后画图均可在此区间进行，以相对坐标来定义位置。如在图 3-6(a)所示的显示方式下，当定义了一个左上角坐标为(200，50)，右下角坐标为(400，150)的一个区域为图视口，则以后处理图形时，就以其左上角为坐标原点(0，0)，右下角为坐标(200，100)的坐标系来定位图形上各点位置。

Turbo C 为用户提供了一个功能很强的画图软件库，它又称为 Borland 图形接口(BGI)，它包括图形库文件(graphics.lib)，图形头文件(graphics.h)和许多图形显示器(图形终端)的驱动程序(如 CGA.BGI、EGAVGA.BGI 等)。还有一些字符集的字体驱动程序(如 goth.chr 黑体字符集等)。编写图形程序时用到的一些图形库函数均在 graphics.lib 中，执行这些函数时，所需的有关信息(如宏定义等)则包含在 graphics.h 头文件中。因此用户在自己的画图源程序中必须包括 graphics.h 头文件，在进行目标程序连接时，要将 graphics.lib 连接到自己的目标程序中去。

由于计算机画图涉及到显示器和驱动它们工作的图形适配器(卡)等许多硬件知识，因而有必要简单地介绍一下。

2.1.1 图形显示器与适配器

计算机中要显示的字符和图形均以数字形式存储在存储器中，而显示器接收的应是模拟信号。插在 PC 微机插槽中的图形卡(即适配器或显卡)，其作用就是将要显示的字符和图形以数字形式存储在卡上的视频存储器 VRAM 中，再将其变成视频模拟信号送往相应适配的显示器进行显示，也即适配器在计算机主机和显示器之间起到了信息转换和视频发送作用，一般 PC 机中适配器、主机、显示器之间的关系如图 3-7 所示。

图 3-7 适配器、主机、显示器之间的关系图



由于计算机配有的显示器种类不同，因而适配器种类不同，而且不同适配器又可支持不同的分辨率显示方式、文本显示方式和颜色设置。表 3-4 提供了 Turbo C 支持的各种显示器适配器和图形模式，其中常用的适配器是下面三种：

1) 彩色图形适配器(CGA)

这是 PC/XT 等微机配用的显示器图形卡，它可以产生单色或彩色字符和图形。在图形方式下，Turbo C 支持两种分辨率供选择：一种为高分辨方式(CGAHI)，象素数为 640×200，这时背景色是黑的(当然也可重新设置)，前景色可供选择，但前景色只是同一种，因而图形只显示两色；另一种为中分辨显示方式，象素数为 320×200，其背景色和前景色均可由用户选择，但仅能显示四种颜色。在该显示方式下，可有四种模式供选择，即 CGACO，CGACl，CGAC2，CGAC3，它们的区别是显示的 4 种颜色不同。

2) 增强型图形适配器〔EGA〕

该适配器与之配接的相应显示器，除支持 CGA 的四种显示模式外，还增加了 Turbo C 称为 EGALO(EGA 低分辨显示方式，分辨率为 640×200)的 16 色显示方式，和 640×350 的 EGAHI（EGA 高分辨显示方式，分辨率为 640×350）的 16 色显示方式。

表 3-4 Turbo C 支持的适配器和图形模式

| 适配器 Driver | 模式 Mode | 分辨率 | 颜色数 | 页数 |
|---|---|---|---|---|
| 标识符 | | | | |
| CGA | 0 | | | |
| | 1 | | | |
| | 2 | | | |
| | 3 | | | |
| | 4 | 320×200 | | |

320×200
320×200
320×200
640×200
4
4
4
4
2
1
1
1
1
1
CGAC0
CGAC1
CGAC2
CGAC3
CGAHI
EGA 0
1
640×200
640×350
16
16
4
2
EGALO
EGAHI
EGA64 0
1
640×200
640×350
16
4
1
1
EGA64LO
EGA64HI
EGAMONO 0 640×350 2 1 EGAMONOHI
VGA 0
1
2
640×200
640×350
640×480
16
16

16
2
2
1
VGALO
VGAMED
VGAHI
MCGA 0
1
2
3
4
5
320×200
320×200
320×200
320×200
640×200
640×480
4
4
4
4
2
2
1
1
1
1
1
1
MCGA0
MCGA1
MCGA2
MCGA3
MCGAMED
MCGAHI
HREC 0 720×348 2 1 HERCMONOHI
ATT400 0
1
2
3
4
5
320×200
320×200
320×200
320×200

640×200
640×400
4
4
4
4
2
2
1
1
1
1
1
1
ATT400C0
ATT400C1
ATT400C2
ATT400C3
ATT400MED
ATT400HI
PC3270 0 720×350 2 1 PC3270HI
IBM8514 0
1
640×480
1024×768
256
256
IBM8514LO
IBM8514HI
26

3) 视频图形阵列适配器(VGA)

它支持 CGA 和 EGA 的所有显示方式，但自己还有 640×480 的高分辨显示方式(VGAHI)、640×350 的中分辨显示方式(VGAMED)和 640×200 的低分辨显示方式(VGALO)，它们均可有 16 种显示颜色可供选择。

众多生产厂家推出了许多性能优于 VGA 但名字各异的图形显示系统，美国标准协会制定了这样的系统应具有的主要性能标准，常将属于这类的显示适配卡统称为 SVGA(即 Super VGA)。目前基本上使用的都属于 SVGA，可以使用 VGA 卡方式进行编程。

显示器的两种工作方式，即文本方式或称字符显示方式和图形显示方式，它们的主要差别是显示存储器(VRAM)中存的信息不同。字符方式时，VRAM 存放要显示字符的 ASCII 码，用它作为地址，取出字符发生器 ROM(固定存储器)中存放的相应字符的图象(又称字模，如图 3-8(a)显示了 C 的 8×8 点阵字模)，变成视频信号在显示器屏上进行显示。EGA、VGA 可以使用几种字符集，如 EGA 下有三种字符集，VGA 有五种字符集。而当选择图形方式时，则要显示的图形的图象直接存在 VRAM 中，VRAM 中某地址单元存放的数就表示了相应屏幕上某行和列上的象素及颜色。图 3-8(b)是字符方式和 CGA320×200 中分辨图形方式显示的示意图，在 CGA 的中分辨图形方式下，每字节代表 4 个象素，即每 2 位表示一个象素及颜色。

上面介绍了一些编制图形程序的预备知识，下面将介绍编制图形程序的各种库函数和相应的其它一些知识。

2.1.2 图形系统的初始化和关闭

在编制图形程序时，进入图形方式前，首先要在程序中对使用的图形系统进行初始化，即要用什么类型的图形显示适配器的驱动程序，采用什么模式的图形方式(也就是相应程序的入口地址)，以及该适配器驱动程序的寻找路径名。所用系统的显示适配器一定要支持所选用的显示模式，否则将出错。当图形系统初始化后，才可进行画图操作。

1) 图形系统的初始化函数

Turbo C 提供了函数 initgraph 可完成图形系统初始化的功能。其原型是：

void far initgraph(int far *driver，int far *mode，char far *path_for_driver)；

当我们使用的存储模式为 tiny(微型)、small(小型)或 medium(中型)时，不需要远指针，因而可以将初始化函数调用格式写成如下形式(该说明适用于后面所述的任一函数)：

initgraph(&graphdriver，&graphmode，"")；

图 3-8 (a)C 的 8×8 点阵字模 (b)VRAM 和字符及图形显示关系



其中驱动程序目录路径为空字符""时，表示就在当前目录下，参数 graphmode 用如表 3-4 所示的模式号或标识符来定义。参数 graphdriver 是一个枚举变量，它属于显示器驱动程序的枚举类型：

enum graphics_driver {DETECT，CGA，MCGA，EGA，EGA64，EGAMONO，IBM 8514，HERCMONO，ATT400，VGA，PC3270}；

其中枚举成员的值顺序为：DETECT 为 0，CGA 为 1，依次类推。当我们不知道所用显示适配器名称时，可将 graphdriver 设成 DETECT，它将自动检测所用显示适配器类型，并将相应的驱动程序装入，并将其最高的显示模式作为当前显示模式，如下面所列：

检测到的适配器 选中的显示模式

CGA 4(640×200，2 色即 CGAHI)

EGA 1(640×350，16 色，即 EGAHI)

VGA 2(640×480，16 色，即 VGAHI)

一旦执行了初始化，显示器即被设置成相应模式的图形方式。

例程 3-9：下面是一般画图程序的开始部分，它包括对图形系统的初始化：

```
#include <graphics.h>
main()
{
int graphdriver=DETECT；
int graphmode；
initgraph(&graphdriver，&grapllmode，"")；
?
}
```

上面初始化过程中，将由 DETECT 检测所用适配器类型，并将当前目录录下相应的驱动程序装入，并采用最高分辨率显示模式作为 graphmode 的值。

若已知所用图形适配器为 VGA 时，想采用 640×480 的高分辨显示模式 VGAHI，则图形初始化部分可写成：

```
int graphdriver=VGA；
int graphmode=VGAHI；
initgraph(&graphdriver，&graphmode，"")；
```

2) 图形系统检测函数

当 graphdriver=DETECT 时，实际上 initgraph 函数又调用了图形系统检测函数 detectgraph，它完成对适配器的检查并得到显示器类型号和相应的最高分辨率模式，若所设适配器不是规定的那些类型，则返回-2，表示适配器不存在，该函数的原型说明是：

void far detectgraph(int far *graphdriver，int far *graphmode)；

例程 3-10：当想检测所用的适配器类型，但并不想用其最高分辨率显示模式，而想由自己

进行控制使用时，可采用这个函数来实现：

```
#include <graphics.h>
main()
{
int graphdriver；
int graphmode；
detectgraph(&graphdriver，&graphmode)；
switch(graphdriver) {
28
case CGA：graphmode=1; /* 设置成低分辨模式 */
break;
case EGA：graphmode=0; /* 设置成低分辨模式 */
break;
case VGA：graphmode=1; /* 设置成中分辨模式 */
break;
case –2:
printf(“\nGraphics adapter not installed”);
exit(1);
default:
printf(“\nGraphics adapter is not CGA，EGA，or VGA”)；
}
initgraph(&graphdriver,&graphmode,””);
?
}
```

调用 detectgraph 时，该函数将把检测到的适配器类型赋予 graphdriver，再把该类型适配器支持的最高分辨率模式赋给 graphmode。

3) 清屏和恢复显示方式的函数

画图前一般需清除屏幕，使得屏幕如同一张白纸，以画最新最美的图画，因而必须使用清屏函数。清屏函数的原型是：

```
void far cleardevice(void)；
```

该函数作用范围为整个屏幕，如果用函数 setviewport 定义一个图视窗口，则可用清除图视口函数，它仅清除图视口区域内的内容，该函数的说明原型是：

```
void far clearviewport(void);
```

当画图程序结束，回到文本方式时，要关闭图形系统，回到文本方式，该函数的说明原型是：

```
void far closegraph(void)；
```

由于进入 C 环境进行编程时，即进入文本方式，因而为了在画图程序结束后恢复原来的最初状况，一般在画图程序结束前调用该函数，使其恢复到文本方式。

为了不关闭图形系统，使相应适配器的驱动程序和字符集(字库)仍驻留在内存，但又回到原来所设置的模式，则可用恢复工作模式函数，它也同时进行清屏操作，它的说明原型是：

```
void far restorecrtmode(void)；
```

该函数常和另一设置图形工作模式函数 setgraphmode 交互使用，使得显示器工作方式在图形和文本方式之间来回切换，这在编制菜单程序和说明程序时很有用处。

2.1.3 基本绘图函数

图形由点、线、面组成，Turbo C 提供了一些函数，以完成这些操作，而所谓面则可由对封闭图形填上颜色来实现。当图形系统初始化后，在此阶段将要进行的画图操作均采用缺省值作为参数的当前值，如画图屏幕为全屏，当前开始画图坐标为(0，0)(又称当前画笔位置，虽然这个笔是无形的)，又如采用画图的背景颜色和前景颜色、图形的填充方式，以及可以

采用的字符集（字库）等均为缺省值。

1) 画点函数

void far putpixel(int x，int y，int color);

该函数表示在指定的 x，y 位置画一点，点的显示颜色由设置的 color 值决定，关于颜



色的设置，将在设置颜色函数中介绍。

int far getpixel(int x，int y)；

该函数与 putpixel()相对应，它得到在(x，y)点位置上的象素的颜色值。

例程 3-11：下面是一个画点的程序，它将在 y=20 的恒定位置上，沿 x 方向从 x=200 开始，连续画两个点(间距为 4 个象素位置)，又间隔 16 个点位置，再画两个点，如此循环，直到 x=300 为止，每画出的两个点中的第一个由 putpixel(x，20，1)所画，第二个则由 putplxel(x+4，20，2)画出，颜色值分别设为 1 和 2。

```
#include <graphics.h>
main()
{
int graphdriver=CGA;
int graphmode=CGAC0,x;
initgraph(&graphdriver,&graphmode,"");
cleardevice();
for(x=20;x<=300;x+=16)
{
putpixel(x,20,1);
putpixel(x+4,20,2);
}
getch();
closegraph();
}
```

为了观查点方便，设置了适配器类型为 CGA，CGAC0 中分辨显示模式。由于 VGA 和它兼容，因此若显示器为 VGA，此时它即以此兼容的仿真形式显示。第一个点显示颜色为绿色，第二个点为红色。只是使用了 VGA，显示的象素点比较小，不大容易在显示器上找到。

2) 有关画图坐标位置的函数

在屏幕上画线时，如同在纸上画线一样。画笔要放在开始画图的位置，并经常要抬笔移动，以便到另一位置再画。我们也可想象在屏上画图时，有一无形的画笔，可以控制它的定位、移动(不画)，也可知道它能移动的最大位置限制等。完成这些功能的函数是：

①移动画笔到指定的(x，y)位置，移动过程不画：

void far moveto(int x，int y)；

②画笔从现行位置(x，y)处移到一位置增量处(x+dx，y+dx)，移动过程不画：

void far moverel(int dx，int dy)；

③得到当前画笔所在位置

int far getx(void)；

得到当前画笔的 x 位置

int far gety(void)；

得到当前画笔的 y 位置

3) 画线函数

这类函数提供了从一个点到另一个点用设定的颜色画一条直线的功能，起始点的设定方法不同，因而有下面不同的画线函数：

①两点之间画线函数。

void far line(int x0，int y0，int x1，int y1)；

从(x0，y0)点到(x1，y1)点画一直线。

②从现行画笔位置到某点画线函数。

void far lineto(int x，int y)；

将从现行画笔位置到(x，y)点画一直线。

③从现行画笔位置到一增量位置画线函数

void far linerel(int dx，int dy)；

将从现行画笔位置(x，y)到位置增量处(x+dx，y+dy)画一直线。

例程 3-12：下面的程序将用 moveto 函数将画笔移到(100，20)处，然后从(100，20)到(100，80)用 1ineto 函数画一直线。再将画笔移到(200，20)处，用 lineto 画一直线到(100，80)处，再用 line 函数在(100，90)到(200，90)间连一直线。接着又从上次 1ineto 画线结束位置开始(它是当前画笔的位置)，即从(100，80)点开始到 x 增量为 0，y 增量为 20 的点(100，100)为止用 linerel 函数画一直线。moverel(-100，0)将使画笔从上次用 1inerel(0，20)画直线时的结束位置(100，100)处开始移到(100-100，100-0)，然后用 linerel(30，20)从(0，100)处再画直线至(0+30，100+20)处。

用 line 函数画直线时，将不考虑画笔位置，它也不影响画笔原来的位置，lineto 和 1inerel 要求画笔位置，画线起点从此位置开始，而结束位置就是画笔画线完后停留的位置，故这两个函数将改变画笔的位置。

```
#include <graphics.h>
main()
{
int graphdriver=VGA;
int graphmode=VGAHI;
initgraph(&graphdriver,&graphmode,"");
cleardevice();
moveto(100,20);
lineto(100,80);
moveto(200,20);
lineto(100,80);
line(100,90,200,90);
linerel(0,20);
moverel(-100,0);
linerel(30,20);
getch();
closegraph();
}
```

4) 画矩形和条形图函数

画矩形函数 rectangle 将画出一个矩形框，而画条形函数 bar 将以给定的填充模式和填充颜色画出一个条形图，而不是一个条形框，关于填充模式和颜色将在后面介绍。

①画矩形函数

void far rectangle(int xl，int y1，int x2，int y2)；

该函数将以(x1，y1)为左上角，(x2，y2)为右下角画一矩形框。



②画条形图函数

void bar(int x1，int y1，int x2，int y2)；

该函数将以(xl，y1)为左上角，(x2，y2)为右下角画一实形条状图，没有边框，图的颜色和填充模式可以设定。若没有设定，则使用缺省模式。

例程 3-13：下面的程序将由 rectangle 函数以(100，20)为左上角，(200，50)为右下角画一矩形，接着又由 bar 函数以(100，80)为左上角，(150，180)为右下角画一实形条状图，用缺省颜色(白色)填充。

```
#include <graphics.h>
main()
{
int graphdriver=DETECT；
int graphmode,x；
initgraph(&graphdriver,&graphmode,"")；
cleardevice()；
rectangle(100，20，200，50)；
bar(100，80，150，180)5
getch()；
closegraph()；
}
```

5) 画椭圆、圆和扇形图函数

在画图的函数中，有关于角的概念。在 Turbo C 中是这样规定的：屏的 x 轴方向为 0 度，当半径从此处逆时针方向旋转时，则依次是 90 度、180 度、270 度，当 360 度时，则和 x 轴正向重合，即旋转了一周。如图 3-9 所示。

α

α

起始角=0

结束角

图 3-9 起始角和终止角

①画椭圆函数

void ellipse(int x，int y，int stangle，int endangel，int xradius，int yradius)；

该函数将以(x，y)为中心，以 xradius 和 yradius 为 x 轴和 y 轴半径，从起始角 stangle 开始到 endangle 角结束，画一椭圆线。当 stangle=0，endangle=360 时，则画出的是一个完整的椭圆，否则画出的将是椭圆弧。关于起始角和终止角规定如图 3-9 所示。

②画圆函数

void far circle(int x，int y，int radius)；

该函数将以(x，y)为圆心，radius 为半径画个圆。

③画圆弧函数

void far arc(int x，int y，int stangle，int endangle，int radius)；

该函数将以(x，y)为圆心，radius 为半径，从 stangle 为起始角开始，到 endangle 为结束角画一圆弧。



④画扇形图函数

void far pieslice(int x，int y，int stangle，int endangle，int radius)；

该函数将以(x，y)为圆心，radius 为半径，从 stangle 为起始角，endangle 为结束角，画一扇形图，扇形图的填充模式和填充颜色可以事先设定，否则以缺省模式进行。

例程 3-14：该程序将用 e11ipse 函数画椭圆，从中心为(320，100)，起始角为 0 度，终止角为 360 度，x 轴半径为 75，y 轴半径为 50 画一椭圆，接着用 circle 函数以(320，220)为圆心，以半径为 50 画圆。然后分别用 pieslice 和 e11ipse 及 arc 函数在下方面出了一扇形图和椭圆弧及圆弧。

```
#include <graphics.h>
main()
{
```

```
int graphdriver=DETECT;
int graphmode,x;
initgraph(&graphdriver,&graphmode,"");
cleardevice();
ellipse(320,100,0,360,75,50);
circle(320,220,50);
pieslice(320,340,30,150,50);
ellipse(320,400,0,180,100,35);
arc(320,400,180,360,50);
getch();
closegraph();
}
```

2.1.4 颜色控制函数

象素的显示颜色，或者说画线、填充面的颜色既可采用缺省值，也可用一些函数来设置。

与文本方式一样，图形方式下，象素也有前景色和背景色。按照 CGA、EGA、VGA 图形适配器的硬件结构，颜色可以通过对其内部相应的寄存器进行编程来改变。

表 3-5 CGA 的调色板号与对应的颜色值

颜色值

模式 调色板号

0 1 2 3

CGAC0 0 背景色 绿 红 黄

CGAC1 1 背景色 青 洋红 白

CGAC2 2 背景色 淡绿 淡红 棕

CGAC3 3 背景色 淡青 淡洋红 淡灰

为了能形象地说明颜色的设置，一般用所谓调色板来进行描述，它实际上对应一些硬件的寄存器。从 C 语言的角度看，调色板就是一张颜色索引表，对 CGA 显示器，在中分辨显示方式下，有 4 种显示模式，每一种模式对应有一个调色板，可用调色板号区别。每个调色板有 4 种颜色可以选择，颜色可以用颜色值 0、1、2、3 来进行选择，由于 CGA 有四个调色板，一旦显示模式确定后，调色板即确定，如选 CGAC0 模式，则选 0 号调色板，但选调色板的哪种颜色则可由用户根据需要从 0、1、2 和 3 中进行选择，表 3-5 就列出了调色板与对应的颜色值。表中若选调色板的颜色值为 0，表示此时选择的颜色和当时的背景色一样。

33

1) 颜色设置函数

①前景色设置函数

void far setcolor(int color)；

该函数将使得前景以所选 color 颜色进行显示，对 CGA，当为中分辨模式时只能选 0，1，2，3。

②选择背景颜色的函数

void far setbkcolor(int color)

该函数将使得背景色按所选 16 种中的一种 color 颜色进行显示，表 3-6 列出了颜色值 color 对应的颜色，此函数使用时，color 既可用值表示，也可用相应的大写颜色名来表示。

表 3-6 背景色值与对应的颜色名

颜色值 颜色名 颜色 颜色值 颜色名 颜色

0 BLACK 黑 8 DARKGRAY 深灰

1 BLUE 蓝 9 LIGHTBLUE 淡蓝

2 GREEN 绿 10 LIGHTGREEN 淡绿

3 CYAN 青 11 LIGHTCYAN 淡青

4 RED 红 12 LIGHTRED 淡红

5 MAGENTA 洋红 13 LIGHTMAGENTA 淡洋红

6 BROWN 棕 14 YELLOW 黄

7 LIGHTGRAY 浅灰 15 WHITE 白

例程 3-15：下面的程序用 initgraph 设置为 CGA 显示器，显示模式为 CGAC0，再用 setcolor 选择显示颜色，由于 color 选为 l，这样图形将选用 0 号调色板的绿色显示，因而用 1ine 函数将画出一条绿色直线来，背景色由于没设置，故为缺省值，即黑色。当按任一键后，执行 setbkcolor，设置背景色为蓝色(也可用 1)，这时用 line 画出的(20，40)到(150，150)的线仍为绿色，但背景色变为蓝色。当再按一键后，程序往下执行，这时用 setcolor 又设显示前景颜色，颜色号为 0，表示画线选背景色，此时用 line(60，120，220，220)画出的线将显不出来，因前景、背景色一样，混为一体。

```
#include <graphics.h>
main()
{
int graphdriver=CGA;
int graphmode=CGAC0;
initgraph(&graphdriver,&graphmode,"");
cleardevice();
setcolor(1);
line(0,0,100,100);
getch();
setbkcolor(BLUE);
line(20,40,150,150);
getch();
setcolor(0);
line(60,120,220,220);
getch();
```



```
closegraph();
}
```

应该提起注意的是，setbkcolor 函数的参数 color 和 setcotor 函数中的 color 不是同一个含义，前者只能选表 3-6 的 16 色之一，而 setcolor 只能选表 3-5 的颜色值，该值对于不同的调色板所表示的颜色不同。可以这样理解，当用 setbkcolor 选了 16 种之一的颜色作背景色后，该颜色就放到调色板的颜色值为 0 处，即改变了调色板颜色值为 0 时代表的颜色。

对于 640×200 高分辨显示模式 CGAHI，颜色只能选 0 或 1，当选其它值时，仍作为 l，即两色显示，当然背景色可选 16 种之一，前景色为白色。若用 EGA 或 VGA 显示器仿真 CGA，上述的说法是正确的。即在上述显示器上选择 CGA 兼容方式，令 graphdriver=CGA，graphmode=CGAHI，是正确的。但用在真正的 CGA 显示器上，如 PC/XT 机的显示器上，设置的背景色被用作前景色，而用 setcolor 设置的前景色却用作背景色，正好相反，这是由于两种显示器硬件结构不同，而 Turbo C 设置函数时，只考虑到 EGA、VGA 的颜色设置正确，没有兼顾到 CGA 硬件结构特点，因此我们使用不同显示器时，对 CGA 方式的 CGAHI 显示模式的编程要注意到这点。

2) 调色板颜色的设置

①调色板颜色的设置函数

```
void far setpalette(int index, int actual_color);
```

该函数用来对调色板进行颜色设置，一般用在 EGA、VGA 显示方式上。

表 3-7 16 个调色板寄存器对应的标准色和值

寄存器号 颜色名 值 寄存器号 颜色名 值

0 EGA_BLACK 0 8 EGA_DARKGRAY 8

1 EGA_BLUE 1 9 EGA_LIGHTBLUE 9

2 EGA_GREEN 2 10 EGA_LIGHTGREEN 10

3 EGA_CYAN 3 11 EGA_LIGHTCYAN 11

4 EGA_RED 4 12 EGA_LIGHTRED 12

5 EGA_MAGENTA 5 13 EGA_LIGHTMAGENTA 13

6 EGA_BROWN 6 14 EGA_YELLOW 14

7 EGA_LIGHTGRAY 7 15 EGA_WHITE 15

对 EGA、VGA 显示器，只有一个调色板，但这个调色板有 l6 个调色板寄存器，它们存的内容对 EGA 和 VGA 含义不同。对 EGA 显示器，调色板是一个颜色索引表，它存有 16 种颜色，VRAM 中的每个象素值(是 4 位)实际上代表一个颜色索引号。由该值即上述函数的参数 index 可知道选中哪个调色板寄存器，而每个调色板寄存器寄存了一种颜色，寄存的颜色可由参数 actual_color 进行设置。由于调色板寄存器为 6 位（如图 3-10 所示），位为 0，则闭断该颜色，为 l 则接通，因而 6 位的组合可产生供参数 actual_color 选择的 64 种颜色。对 EGA 图形系统初始化时，16 个调色板寄存器已装入确定的颜色，也称为标准色。显示模式不同，所装颜色值也不同，表 3-7 是指 EGAHI 或 VGAHI 模式下的标准色。例如，若 VRAM 中 3 个象素值分别为 1001、0000 和 1111，即 index 值(或调色板寄存器号)分别为 9、0 和 15，则这三个象素点在显示屏上分别显示亮蓝、黑和白，实际上第二个总不显示(因和背景色同，前题是，背景色也是黑的)。

35

X X R' G' B' R G B

淡红 淡绿 淡蓝 红 绿 蓝

D7 D6 D5 D4 D3 D2 D1 D0

图 3-10 调色板寄存器六位数代表的颜色

当编制动画或菜单等高级程序时，系统图形初始化时常需要改变每个调色板寄存器的颜色设置，这时就可用 setpalette 函数来重新对某一个调色板寄存器颜色进行再设置。

对于 VGA 显示器，也只有一个调色板，对应 16 个调色板寄存器。但这些寄存器装的内容和 EGA 的不同，它们装的又是一个颜色寄存器表的索引。共有 256 个颜色寄存器供索引。VGA 的调色板寄存器是 6 位，而要寻址 256 个颜色寄存器需有 8 位，所以还要通过一个被称为模式控制寄存器的最高位(即第 7 位)的值来决定：若为 0(对于 640×480×16 色显示是这样)，则低 6 位由调色板寄存器来给出，高两位由颜色选择寄存器给出，从而组合出 8 位地址码。因此它的象素显示过程是：由 VRAM 提供调色板寄存器索引号(0~15)，再由检索到的调色板寄存器的内容同颜色选择寄存器配合，检索到颜色寄存器，再由颜色寄存器存的颜色值而令显示器显示；当模式寄存器最高位为 1 时，则调色板寄存器给出低 4 位的 4 位地址码，而由颜色选择寄存器给出高 4 位的 4 位地址码，来组合成 8 位地址码对颜色寄存器寻址而得出颜色值。这里的调色板寄存器，颜色选择寄存器，模式控制寄存器和颜色寄存器均属于 VGA 显示器中的属性控制器。

由于 Turbo C 中没有支持 VGA 的 256 色的图形模式，只有 16 色方式，因而 16 个颜色寄存器寄存了 16 个颜色寄存器索引号，它们代表的颜色如表 3-7 所列，所显示的颜色和 CGA 下选背景色的顺序一样。EGA 和 VGA 的调色板寄存器装的值虽然一样(当图形系统初始化时，指缺省值)，但含义不同，前者装的是颜色值，后者装的是颜色寄存器索引号，不过它们最终表示的颜色是一致的，因而当用 setpalette(index actual_color)对 index 指出的某个调色板寄存器重新设置颜色时，actual_color 可用表 3-7 所指的颜色值，也可用大写名，如 EGA_BLACK，EGA_BLUE 等。在缺省情况下，和 CGA 上 16 色顺序一样，当使用 setpalette 函数时，index 只能取 0~15，而 actual_color 若其值是表 3-7 所列的值，则调色板颜色保持不变，即调色板寄存器值不变。

②改变调色板 16 种颜色的函数

void far setallpalette(struct palettetype far *palette);

其中结构 palettetype 定义如下：

#define MAXCOLORS l5
struct palattetype {
unsigned char size;
signed char colors[MAXCOLORS+1];
};
该定义在头文件 graphics.h 中。size 元素由适配器类型和当前模式下调色板的颜色数决定，即调色板寄存器数。colors 是个数组，它实际上代表调色板寄存器，每个数组元素的值就表示相应调色板寄存器的颜色值。对 VGA 的 VGAHI 模式，size=16，缺省的 colors 的各元素值就相当于表 3-7 所列值。
③得到调色板颜色数和颜色值的函数
与上述两个函数对应的是如下两个函数：
void far getpalette(struct palettetype far *palette);
void far getpalettesize(void);

前者将得到调色板的颜色数(即调色板寄存器个数)和装的颜色值，后者将得出调色板颜色数。getpalette 函数将把得到信息存入由 palette 指向的结构中，其结构 palettetype 定义如上所述。
例程 3-16：下面的一个程序演示了调色板颜色设置函数的作用，首先程序用 setcolor(1)，设置了前景颜色，其中参数 1 表示用 1 号调色板寄存器中的颜色，即缺省值为蓝色，这样用 rectangle 将画出一个蓝色的方框，然后按任一键，这时用 setpalette(1，i)函数将分别设置一号调色板寄存器为绿、青、红、黄、亮白。每按一键，方框颜色改变一次，直到方框变成亮白为止。因为调色板相应的调色板寄存器所装的颜色一旦改变．用 setcolor(调色板寄存器号)设置的颜色也立即改变，可以这样形象地理解，setcolor 函数相当于把代表红、绿、蓝的三条模拟电压线接通，由于它们电压高低不尽相同，所以显示器混合出来的颜色也不相同，当调成蓝色时，三条模拟电压线代表红绿的电压近似为零，因而这时方框便是蓝色的。然而我们用函数 setpallette 再调三条模拟电压线的电压比例(如同用电位器调一样)，因而原来画好的框，其颜色将跟着立即改变。
该程序中将 palette 定义成 palettetype 类型结构，palettetype 结构如前所述在 graphics.h 头文件中已有定义，这样用 getpalette(&pa11ette)函数得到图形系统初始化的各调色板寄存器的颜色值(即缺省值)，然后在程序快结束时，用 setallpalette(&palette)恢复各调色板寄存器的原来值。
可以做实验，将第一个调色板寄存器的颜色值分别置成 6，8，9，?，15 时产生的颜色并不是标准的 BROWN、DARKGRAY、LIGHTBLUE，即和置成 20，56，57?63 的颜色不一样，这也证明了在 EGA/VGA l6 色显示图形方式下，16 个调色板寄存器装的颜色值(缺省值)不同于 CGA 的 16 色值，但其对应的颜色却是一致的，即两者代表同一颜色的颜色值不同，可参阅表 3-7。
#include <graphics.h>
main()
{
int graphdriver=DETECT,graphmode;
struct palettetype palette;
int i,j;
initgraph(&graphdriver,&graphmode,"");
getpalette(&palette);
setcolor(1);
rectangle(200,200,300,320);
getch();
i=2;j=2;

```
do
{
printf("color=%d",j);
setpalette(1,i);
getch();
i++; j++;
if(i==6) i=20;
if(i==21) i=7;
if(i==8) i=56;
} while (j<16);
```

37

```
getch();
setallpalette(&palette);
closegraph();
}
```

2.1.5 画线的线型函数

1) 设定线型函数

在前述的例子中画线、画圆、画框时，线的宽度都是一样的，实际上 Turbo C 也提供了改变线的宽度、类型的函数：

void far setlinestyle(int linestyle，unsigned upattern，int thickness)；

当线的宽度参数(thickness)不设定时，取缺省值，即一个象素宽，当设定为 3 时，可取三个象素宽，取值见表 3-8。当线型参数(1inestyle)不设定时，取缺省值，即实线；设定时，可有 5 种选择如表 3-9 所列。upattern 参数只有在 1inestyle 取 4 或 USERBIT_LINE 时才有意义，即表示在用户自定义线型时，该参数才有用。该参数若表示成 16 位二进制数，则每位代表一个象素。是 1 的位，代表的象素用前景色显示，是 0 的位，代表的象素用背景色显示(实际没有显示)，例如图 3-11 表示了由 16 个象素构成的一个 16 个象素长的线段，线宽为 1 个象素宽。当 1ineseyle 不是 USERBIT_LINE 时，upattern 取 0 值。

表 3-8 线宽(thickness)

符号名 值 含义

NORM_WIDTH 1 一个象素宽

THICK_WIDTH 3 三个象素宽

表 3-9 直线的形状(linestyle)

符号名 值 含义

SOLID_LINE 0 实线

DOTTED_LINE 1 点线

CENTER_LINE 2 中心线

DASHED_LINE 3 点画线

USERBIT_LINE 4 用户自定义线

F 3 D 0

图 3-11 用 setlinestyle(4，0xF3D0，1)设置的线型

例程 3-17：下面的程序首先在屏中间以屏中心为圆心，半径为 98 画出一个绿色的圆框，由于没有设置画线的线型和线宽，故取缺省值为 1 个象素宽的实线。接着用 setcolor(12)设置前景色为淡红色。程序进入 for 循环，而画出线宽交替为 1 个和 3 个象素宽的 15 个矩形框来，框由小到大，一个套一个，颜色为淡红色。程序的下一个 for 循环将用 2、3、4 和 5 颜色，即用绿、青、红、洋红分别画出通过屏幕中心的 4 条线，线型分别是实线、点线、中心线和点划线，线宽为 3 个象素，如此重复，共画出 5 组。

程序最后用 setcolor(EGA_WHITE)设置画线颜色为白色，将用自定线型(0x1001)在原先画出的绿色圆框中，标出一个十字线，线的形状为 4 个白点，8 个不显示点，又 4 个白点，

接着又重复这个线段模式，直到画至圆周上而终止。由于人的视觉分辨能力，我们将 4 个白点看成了一个点，因此出现了屏幕上的那种现象。

```
#include <graphics.h>
main()
{
int graphdriver=VGA,graphmode =VGAHI;
int i,j,x1,y1,x2,y2;
initgraph(&graphdriver,&graphmode,""),
setbkcolor(EGA_BLUE);
cleardevice();
setcolor(EGA_GREEN);
circle(320,240,98); /* 画出一个绿色圆 */
setcolor(12); /* 设置颜色为淡红色 */
j=0;
for(i=0;i<=90;i=i+6)
{
setlinestyle(0,0,j); /* 画出一个套一个的矩形框 */
x1=440-i;y1=280-i;
x2=440+i;y2=280+i;
rectangle(x1,y1,x2,y2);
j=j+3;
if(j>4) j=0;
}
j=0;
for ( i=0;i<=180;i=i+16) /* 画出通过屏幕中心的 4 种线型的 4 色线 */
{
if(j>3)j=0;
setcolor(j+2);
setlinestyle(j,0,3);
j++;
x1=0;y1=i,
x2=640;y2=480-i;
line(x1,y1,x2,y2);
}
setcolor(EGA_WHITE);
setlinestyle(4,0x1001,1); /* 用户定义线型,1 个象素宽 */
line(220,240,420,240); /* 画出通过圆心的 y 线 */
line(320,140,320,340); /* 画出通过圆心的 x 线 */
getch();
closegraph();
}
```

2) 得到当前画线信息的函数

与设定线型函数 setlinestyle 相对应的是得到当前有关线的信息的函数：



```
void far getlinesettings(struct linesettingstype far *lineinfo);
```

该函数将把当前有关线的信息存放到由 lineinfo 指向的结构中，其结构 linesetingstype 定义如下：

structlinesettingstype {

int linestyle；

unsigned upattern；

int thickness；

};

2.1.6 封闭图形的填色函数及有关画图函数

Turbo C 提供了一些画基本图形的函数，如我们前面介绍过的画条形图函数 bar 和将要介绍的一些函数，它们首先画出一个封闭的轮廓，然后再按设定的颜色和模式进行填充，设定颜色和模式有特定的函数。

1) 填色函数

void far setfilestyle(int pattern，int color)；

该函数将用设定的 color 颜色和 pattern 图模式对后面画出的轮廓图进行填充，这些图轮廓是由待定函数画出的，color 实际上就是调色板寄存器索引号，对 VGAHI 方式为 0~l5，即 l6 色，pattern 表示填充模式，可用表 3-10 中的值或符号名表示。

表 3-10 填充模式(pattern)的规定

符号名 值 含义

EMPTY_FILL 0 用背景色填充

SOLID_FILL 1 用单色实填充

LINE_FILL 2 用"一"线填充

LTSLASH_FILL 3 用"//"线填充

SLASH_FILL 4 用粗"//"线填充

BKSLASH_FILL 5 用"\\"线填充

LTBKSLASH_FILL 6 用粗"\\"线填充

HATCH_FILL 7 用方网格线填充

XHATCH_FILL 8 用斜网格线填充

INTTERLEAVE_FILL 9 用间隔点填充

WIDE_DOT_FILL 10 用稀疏点填充

CLOSE_DOT_FILL 11 用密集点填充

USER_FILL 12 用用户定义样式填充

当 pattern 选用 USER_FILL 用户自定义样式填充时，setfillstyle 函数对填充的模式和颜色不起任何作用，若要选用 USER_FILL 样式填充时，可选用下面的函数。

2) 用户自定义填充函数

void far setfillpattern(char *upattefn，int color)；

该函数设置用户自定义可填充模式，以 color 指出的颜色对封闭图形进行填充。这里的 color 实际上就是调色板寄存器号，也可用颜色名代替。参数 upattern 是一个指向 8 个字节存储区的指针，这 8 个字节表示了一个 8×8 象素点阵组成的填充图模，它是由用户自定义的，它将用来对封闭图形填充。8 个字节的图模是这样形成的：每个字节代表一行，而每个



字节的每一个二进制位代表该行的对应列上的象素。是 1，则用 color 显示，是 0 则不显示。

例程 3-18：下面的程序演示了如何用 setfillstyle(SOLID_FILL，color)对用 bar 生成的条状图进行填充。对 VGA 显示器，由于可显示 16 色，因而通过 getpalette(&palette)函数，可得出 palette，size 为 16。这样 for 循环，将使得用 bar 函数生成的 l6 个小条分别填充上调色板上的 16 种颜色，其顺序为缺省时(即标准的)调色板各寄存器的顺序颜色。

do 循环又利用随机函数 random(palette.size)随机产生 0~palette.size 的整数，对调色板各寄存器的颜色重新进行设置，这样一旦 setpalene 函数对某调色板寄存器进行了颜色的重新设置，则代表相应号调色板寄存器的小条颜色立即变成新设的颜色。若随机遇到 random 产生一个 0，则 setpalette 立即对 0 号调色板用第二次 random 产生的数进行颜色设置，因而此时整个屏幕显示的背景色立即发生变化，颜色为第二次 random 产生的颜色。关于调色板的

设置问题，可参阅前面已介绍过的调色板设置函数。

```
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
main()
{
int graphdriver=DETECT，graphmode；
struct palettetype palette；
int color；
initgraph(&graphdriver，&graphmode,"");
getpalette(&palette)；
for (color=0;color<palette.size;color++ ) /* 对 16 个小条用 16 种颜色填充 */
{
setfillstyle(SOLID_FILL，color)；
bar(20*(color-1)，0，20*color，20);
}
if(palette.size>1)
{
do /* 对调色板 16 种颜色重新进行设置 */
setpalette(random(palette.size),random(palette.size));
while ( !kbhit() );
getch();
}
setallpalette(&palette)；
closegraph();
}
```

例程 3-19：下面的程序演示了用不同填充图模(pattern)对由 bar 和 pieslice 函数产生的条状和扇形图进行颜色填充。运行程序，可以看出第 1 个 bar(0，0，100，100)产生的方条将由蓝色的斜线填充，即以 LTSlASH_FILL(3)图模填充。接着将由红色的网格(HATCH_FILL，RED)图模填充一个扇形。由于缺省时，前景颜色为白色，故该扇形将用白色边框画出，接着用户自定义填充模式，因而用 bar(100，100，200，200)画出的方条，将用用户定义的图模(用字符数组 gray50[]表示的图模)，用黄色进行填充。

```
#include <graphics.h>
main()
41
{
int graphdriver=VGA,graphmode=VGAHI;
struct fillsettingstype save;
char savepattern[8];
char gray50[]={0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x81};
initgraph(&graphdriver,&graphmode,"");
getfillsettings(&save); /* 得到初始化时填充模式 */
if(save.pattern != USER_FILL )
setfillstyle(3,BLUE);
bar(0,0,100,100);
setfillstyle(HATCH_FILL,RED);
pieslice(200,300,90,180,90);
setfillpattern(gray50,YELLOW); /* 设定用户自定义图模进行填充*/
```

```
bar(100,100,200,200);
if(save.pattern==USER_FILL)
setfillpattern(savepattern,save.color);
else
setfillpattern(savepattern, save.color); /* 恢复原来的填充模式 */
getch();
closegraph();
}
```

3) 得到填充模式和颜色的函数

```
void far fillsettings(struct fillsettingstype far *fillinfo);
```

它将得到当前的填充模式和颜色，这些信息存在结构指针变量 fillinfo 指出的结构中该结构定义是：

```
struct fillsettingstype
{
int pattern； /* 当前填充模式 */
int color; /* 填充颜色 */
};
```

```
void far getfillpattern(char *upattern)；
```

该函数将把用户自定义的填充模式和颜色存入由 upattern 指向的内存区域中。

4) 与填充函数有关的作图函数

前面我们已经介绍了画条形图函数 bar 和画扇形函数 pieslise，它们需要用 setfillstyle 函数设置填充模式和颜色，否则按缺省方式。另外还有一些画图形的函数，也要用到填充函数。

①画三维立体直方图函数

```
void far bar3d(int x1，int y1，int x2，int y2，int depth，int topflag)；
```

该函数参数名定义如图 3-12 所示。当 topflag 非 0 时，画出三维顶，否则将不画出三维顶，depth 决定了三维直方图的长度。

②画椭圆扇形函数

```
viod far sector(int x，int y，int stangle，int endang1e，int xradius，int yradius)；
```

该函数将以(x，y)为圆心，以 xradius 和 yradius 为 x 轴和 y 轴半径，从起始角 stang1e 开始到 endang1e 角结束，画一椭圆扇形图，并按设置的填充模式和颜色填充。当 stang1e 为



0，endangle 为 360 时，则画出一完整的椭圆图。

(x1,y1)

(x2,y2)

depth

topflag==1 topflag==0

depth

图 3-12 画三维立体直方图函数的参数名定义

③画椭圆图函数

```
void far fillellipse(int x，int y，int xradius，int yradius)； ”
```

该函数将以(x，y)为圆心，以 xradius 和 yradius 为 x 轴和 y 轴半径，画一椭圆图，并以设定或缺省模式和颜色填充。

④画多边形图函数

```
void far fillpoly(int numpoints，int far *polypoints)
```

该函数将画出一个顶点数为 numpoints，各顶点坐标由 polypoints 给出的多边形，也即边数为 polypoints-1，当为一封闭图形时，numpohts 应为多边形的顶点数加 1，并且第一个顶点坐标应和最后一个顶点的坐标相同。

例程 3-20：下面程序用 bar3d 函数画出了一个立方图，并且画面用蓝色斜线填充，接着由第

二个 bar3d 函数又在相邻位置画出一个没有顶的三维图，画面用红色方格填充。该函数的 topflag=0。在屏幕下方，由 sector 函数画出了一个不完整的椭圆，并用绿色填充，可以看出差 120 度就是一个完整的椭圆了。在其相邻位置则是由航 lellipse 函数画出的一个椭园，它用谈红色填充，屏幕的右上半是由捌 1p01y 函数画出的一个六边图形，被填以洋红色，由于最初顶点坐标和最后一个顶点坐标相同(同为(420，20))，所以是一个封闭的图形。

```
#include <graphics.h>
main()
{
int graphdriver=VGA,graphmode=VGAHI;
struct fillsettingstype save;
char savepattern[8];
int d[]={420,20,330,45,330,145,420,120,510,145,510,55,420,20};
initgraph(&graphdriver,&graphmode,"");
getfillsettings(&save);
setfillstyle(3,BLUE);
bar3d(100,50,150,120,30,1);
setfillstyle(HATCH_FILL,RED);
bar3d(200,50,250,120,30,0);
setfillstyle(1,GREEN);
sector(200,300,0,250,100,40);
setfillstyle(1,LIGHTRED);
43
fillellipse(420,300,100,40);
setfillstyle(1,5);
fillpoly(7,d);
getch();
setfillstyle(save.pattern,save.color);
closegraph();
}
```

5) 可对任意封闭图形填充的函数

前面介绍的填充函数，只能对由上述特定函数产生的图形进行颜色填充，对任意封闭图形均可进行填充的还有一函数，其原型说明为：

void far floodfill(int x，int y，int border)；

该函数将对一封闭图形进行填充，其颜色和模式将由设定的或缺省的图模与颜色决定。其中参数(x，y)为封闭图形中的任一点，border 是封闭图形的边框颜色。编程时该函数位于画图形的函数之后，即要填充该图形。需要注意的是：

a)若(x，y)点位于封闭图形边界上，该函数将不进行填充。

b)若对不是封闭的图形进行填充，则会填到别的地方，即会溢出。

c)若(x，y)点在封闭图形之外，将对封闭图形外进行填充。

d)由参数 border 指出的颜色必须与封闭图形的轮廓线的颜色一致，否则会填到别的地方去。

例程 3-21：下面的程序首先用白色线画出一个长方体，并用设定的亮红色(LIGHTRED)和实填充模式填充该长方体的正面，然后使用两个 floodfill 函数用同样的模式和颜色填充该长方体能看得见的另两面，然后将画线颜色由 setcolor(LIGHTGREEN)设置为亮绿色，并由 setfillstyle 设置填充模式为棕色和用平直线填充，由于该函数对 rectangle 画出的矩形框不起作用，所以并不执行填充，当画好矩形框后，其后的 floodfill 函数将完成对该矩形框的填充，即以棕色的平直线进行填充。可以作实验，当将 floodfill 中的 x，y 参数设在被填图形框外时，结果会将该框外的所有区域填充。当 floodfill 中的 border 指定的颜色和画图框的颜色不

符时，也会将颜色填到图外边去。

```
#include <graphics.h>
main()
{
int graphdriver=VGA，graphmode=VGAHI；
initgraph(&graphdriver，&graphmode，””)；
setbkcolor(BLUE);
setcolor(WHITE)； /* 用白色画线 */
setfillstyle(1，LIGHTRED)； /* 设填充模式和颜色 */
bar3d(100，200，400，350，100，1)； /* 画长方体并填正面 */
floodfill(450，300，WHITE); /* 填侧面 */
floodfill(250，150，WHITE)； /* 填顶部 */
setcolor(LIGHTGREEN);
setfillstyle(2，BROWN)；
rectangle(450，400，500，450); /* 画矩形 */
floodfill(470，420，LIGHTGREEN)； /* 填矩形 */
getch()；
44
closegraph();
}
```

2.1.7 图视口操作函数

1) 图视口设置函数

在图形方式下可以在屏幕上某一区域设置一个窗口，这样以后的画图操作均在这个窗口内进行，且使用的坐标以此窗口顶左上角为(0，0)作参考，而不再用物理屏幕坐标(屏左角为(0，0)点)。在图视口内画的图形将显示出来，超出图视口的部分可以不让其显示出来，也可以让其显示出来(不剪断)，该函数原型说明为：

```
void far setviewport(int xl，int y1，int x2，int y2, clipflag);
```

其中(x1，y1)为图视口的左上角坐标，(x2，y2)为所设置的图视口右下角坐标，它们都是以原屏幕物理坐标为参考的。clipflag 参数若为非 0，则所画图形超出图视口的部分将被切除而不显示出来。若 clipflag 为 0，则超出图视口的图形部分仍将显示出来。

2) 图视口清除与取信息函数

①图视口清除函数

```
void far clearviewport(void)；
```

该函数将清除图视口内的图象。

②取图视口信息函数

```
void far getviewsettings(struct viewport type far *viewport)；
```

该函数将取得当前设置的图视口的信息，它存于由结构 viewporttype 定义的结构变量 viewport 中，结构 viewporttype 定义如下：

```
struct viewporttype {
int left，top，right，bottom;
int clipflag；
}；
```

使用图视口设置函数 setviewport，可以在屏上设置不同的图视口——窗口，甚至部分可以重叠，然而最近一次设置的窗口才是当前窗口，后面的图形操作都视为在此窗口中进行，其它窗口均无效。若不清除那些窗口的内容，则它们仍在屏上保持，当要对它们处理时，可再一次设置那个窗口一次，这样它就又变成当前窗口了。

使用 setbkcolor 设置背景色时，对整个屏幕背景起作用，它不能只改变图视口内的背景，在用 setcolor 设置前景色时，它对图视口内画图起作用。若下一次设置纳图视口没有设置颜

色，那么上次在另一图视口内设置的颜色在本次设置的图视口内仍起作用。

例程 3-22：下面程序首先用 setviewpon 函数设置了一个左上角(0，0)，右下角为(639，199)的图视窗口，并设置 clipflag 参数为 1，即超出图视口的图形将被切除，接着画了一个方框和一个边与方框一边相切的圆，它将完整地显示出来。按任意键后，开一个图视口，用棕色画了和第一个窗口相同的方框和圆，超出图视口的部分被剪切。由于在开此窗口前，没有用 clearviewport()清除上次的窗口内容，所以上次画的洋红色的方框和圆仍保留，当再按任一键后，调用了 clearviewport()函数，因而将窗口中的棕色方框和圆清除了。接着又建立了一个图视窗口，该窗口(50，50，200，l 25)和最初窗口(0，0，639，199)有部分重合，因而重合部分的内容将在当前窗口中显示出来。新画的方框和圆也显示出来(窗外部分内容仍保留)，由于选择了 clipflag=1，所以超出窗口的方框和圆的部分被剪切，不显示出来，但再按任一键后，由于用了 c1earviewport，所以窗口中内容被清除并又在同一位置开辟了同样大小的窗口，但由于 clipflag=0，所以超出的部分没被剪掉，因而可看见。

```
#include <graphics.h>
main()
45
{
int i,graphdriver,graphmode,size,page;
graphdriver=VGA;
graphmode=VGAHI;
initgraph(&graphdriver,&graphmode,"");
cleardevice();
setviewport(0,0,639,199,1); /* 设图视口 */
setcolor(5);
rectangle(50,50,125,100);
circle(100,75,50);
..... GETCH();
setviewport(150,150,639,239,1); /* 又设一图视口 */
setcolor(6);
rectangle(50,50,125,100); /* 方框超出部分被切掉 */
circle(100,75,50);
getch();
clearviewport(); /* 清窗口 */
setviewport(50,50,200,125,1);
rectangle(50,50,125,100);
circle(100,75,50);
getch();
clearviewport();
setviewport(50,50,200,125,0); /* 开窗口,超出部分不切除 */
rectangle(50,50,125,100);
circle(100,75,50);
getch();
clearviewport();
closegraph();
}
```

2.1.8 图形方式下的文本输出函数

在图形方式下，虽然也可以用 printf()，puts()，putchar()函数输出文本，但只能在屏上用白色显示，无法选择输出的颜色，尤其想在屏上定位输出文本，更是困难，且输出格式也是不能变的 80 列×25 行形式。Turbo C 提供了一些专门用在图形方式下的文本输出函数，

它们可以用来选择输出位置，输出字型、大小，输出方向等。

为在图形方式下输出文本，Turbo C 提供一个 8×8 点阵的字库，它们是英文字母和一些常用符号的字模（参见图 3-8(a)），是用 8×8 点阵表示出其图象的字形库。该库嵌入在图形系统中，我们一旦在 Turbo C 下对系统进行了图形系统初始化(即 initgraph())，该字库即被调入内存，这种字库也是在缺省情况下，图形方式中输出文本时采用的字形。另外 Turbo C 的图形接口软件(BGI)还提供 4 种向量字库，又称笔划字库。该字库中，字符用一组向量表示，这些向量表示如何画字符，4 个向量字库在磁盘上用后缀为.chr 文件名存放，它们是 itrip.chr(三倍笔划体字库)、litt.chr(小笔划字库)，sans.chr(无衬笔划字库)，goth.chr(黑体笔划字库)。当文本输出选择这些字库中的一个时，如用 settextstyle()函数，则相应的笔划字库就



被调入内存。

1) 文本输出函数

① 当前位置文本输出函数

void far outtext(char far *textstring)；

该函数将在当前位置在屏上输出由字符串指针 textsering 指出的文本字符串。该函数没有定位参数，只能在当前位置输出字符串。

② 定位文本输出函数

void far outtextxy(int x，int y，char far *textstring)；

该函数将在指定的(x，y)位置输出字符串。(x，y)位置如何确定，还需要用位置确定函数 settextjustify()来确定；选用何种字形显示、字体大小及横向或纵向显示，还需用 settextstyle()函数来确定。这些均要在文本输出函数之前确定。若没有使用函数确定，则输出来用缺省方式，即字形采用 8×8 点阵字库，横向输出，其(x，y)位置表示输出字符串的第一个字符的左上角位置，字体 1：1。例如，当执行函数 outtextxy(10，10，”Turbo C”)；时，Turbo C 将采用缺省方式显示在如图 3-13 所示位置，其“T”字的左上角位置为(10，10)，字形为 8×8 点阵(8 个象素宽，8 个象素高)，尺寸 1：1，即和字库中的字同大。

CRT 显示器

Turbo c

(10,10)

(10,18)

图 3-13 outtextxy(10，10，”Turbo c”)输出

③ 文本输出位置函数

void far settextjustify(int horiz，int vert);

该函数将确定输出字位串时，如何定位(x，y)。即当用 outtext(x，y，”字符串”)或 outtextxy(x，y,”字符串”)输出字符串时，(x，y)点是定位在字符串的哪个位置，horiz 将决定(x，y)点的水平位置相对于输出字符串如何确定，vert 参数将决定(x，y)点的垂直位置相对于输出字符串如何确定。这两个参数的取值和相应的符号名如表 3-11 所示，如若 horiz 取 LEFT_TEXT(或取 0)，则(x，y)点是以输出的第一个字符的左边为开始位置，即(x，y)定位于此。但是以第一个字符左边的顶部、中部，还是底部定位〔x，y〕，还不能确定，也就是说，(x，y)点是指输出字符串第一个字符的左边位置，但是在第一个字符左边的垂直方向上的位置还须由 vert 参数决定。如 vert 取 TOP_TEXT(即 2)，则(x，y)在垂直方向定位于第一个字符左边位置的顶部。

表 3-11 参数 horiz、vert 的取值

参数 horiz 参数 vert

符号名 值 含义 符号名 值 含义

LEFT_TEXT 0 输出左对齐 BOTTOM_TEXT 0 底部对齐

CENTER_TEXT 1 输出以字串中心对齐 CENTER_TEXT 1 中心对齐

RIGHT_TEXT 2 输出右对齐 TOP_TEXT 2 顶部对齐

2) 定义文本字型函数

void far settextstyle(int font，int direction，int char size)；

47

该函数用来设置文本输出的字形、方向和大小，其相应参数 font、参数 direction 和参数 size 的取值如表 3-12 所列。

表 3-12 font、direction、size 的取值

| 符号名 | 值 | 含义 |
| --- | --- | --- |
| DEFAULT_FONT | 0 | 8×8 字符点阵(缺省值) |
| TRIPLEX_FONT | 1 | 三倍笔划体字 |
| SMALL_FONT | 2 | 小字笔划体字 |
| SANS_SERIF_FONT | 3 | 无衬线笔划体字 |
| font | | |
| GOTHIC_FONT | 4 | 黑体笔划体字 |
| 符号名 | 值 | 含义 |
| HORIZ_DIR | 0 | 水平输出 |
| direction | | |
| VERT_DIR | 1 | 垂直输出 |
| 1 | | 8×8 点阵 |
| 2 l | | 6×16 点阵 |
| 3 | | 24×24 点阵 |
| 4 | | 32×32 点阵 |
| 5 | | 40×40 点阵 |
| 6 | | 48×48 点阵 |
| 7 | | 56×56 点阵 |
| 8 | | 64×64 点阵 |
| 9 | | 72×72 点阵 |
| 10 | | 80×80 点阵 |
| size | | |
| USER_CHAR_SIZE | | 用户自定义字符大小 |

例程 3-23：下面的程序用 settextstyle()函数，按表 3-12 所列的 font 参数可取值，分别在屏上不同位置输出了用该字体符号名作为字符串的不同字形，输出方向为水平输出(即 direction 取 HORIZ_DIR)，而输出字符的点阵，即 size 参数则取 2，为 16×16 点阵。程序输出如图 3-14 所示。

```
#include <graphics.h>
main()
{
int i,graphdriver,graphmode,size,page;
char s[30];
graphdriver=DETECT;
initgraph(&graphdriver,&graphmode,"");
cleardevice();
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
settextjustify(LEFT_TEXT,0);
outtextxy(220,20,"Defaut font");
settextstyle(TRIPLEX_FONT,HORIZ_DIR, 2);
settextjustify(LEFT_TEXT,0);
outtextxy(220,50,"Triplex font");
settextstyle(SMALL_FONT,HORIZ_DIR, 2);
settextjustify(LEFT_TEXT,0);
```

48

```
outtextxy(220,80,"Smallfont");
settextstyle(SANS_SERIF_FONT,HORIZ_DIR,2);
settextjustify(LEFT_TEXT,0);
outtextxy(220,110,"Sans serif font");
settextstyle(GOTHIC_FONT,HORIZ_DIR, 2);
settextjustify(LEFT_TEXT,0);
outtextxy(220,140,"gothic font");
getch();
closegraph();
}
```

图 3-14 font 可选字体例 图 3-15 例程 3-24 的屏幕输出

3) 文本输出字符串函数

int sprintf(char *string，char *format[, argument，…]);

该函数将把变量值 argument，按 format 指定的格式，输出到由指针 string 指定的字符串中去，该字符串就代表了其输出。这个函数虽然不是图形专用函数，但它在图形方式下的文本输出中很有用，因为用 outtext()或 outtextxy()函数输出时，输出量是文本字符串，当我们要输出数值时不太方便。因而可用 sprintf 函数将数值输出到一个字符数组中，再让文本输出函数输出这个字符数组中的字符串即可。

例程 3-24：下面的程序首先用 setviewport 开了一个对角坐标为(40，40)和(600，440)的图视窗口，其后的图形操作则在此窗口中进行，超过图视口的部分将被剪切，接着用黄色画了一个矩形框，并用绿色填充。下面的 rectangle()函数又在其框内再画一个矩形框，并用淡洋红色填充，这样，就在屏上出现了带有绿色边框(用黄线勾出了边框四周)的一矩形。然后在其内用蓝色写出”Welcome your”字样，由于用函数 senextstyle(1，0，6)设置使用字体为 TRIPLEX_FONT 的笔划字形，且水平放置(direction=0)，字体大小取 6 倍(char size=6)。因而 welcom your 字符串字形将不同于我们通常看到的印刷字体。接着又用 setviewport(100，200，540，380，0)开辟了又一个窗口，且以后操作时窗口外的图形将不被切除(这种方法常用于调正显示图形在图视口中的位置，以知道显示图形超出了图视口多大距离，便于调正)。由于又设置了一个图视窗口，因而先前建立的那个图视口就不起作用了(由于没使用 clearviewport()函数，所以原窗口内的内容仍保留)，又以此窗口为坐标系，在此内画了一个黄色边框的矩形框。由于用 setfillstyle(1，12)进行了设置，因而其内用淡红色进行了填充，接着用 sprintf 函数将要显示的字符串存在 s[]字符数组内(因要多次用到该字符串，为了方便，故这样操作)，执行完函数 outtextxy(60，40，s)后，又用 9 倍的小号笔划体(settextstyle(2，0，9)的设置)，将”Let’s study Turbo C”显示在淡红色框内，接着用 settextstyle(4，0，3)设置，将该字符串用黑体笔划字放大 3 倍用蓝色显示在该框的下边部分。屏幕输出如图 3-15 所示。

```
#include <graphics.h>
```

49

```
main()
{
int i, graphdriver,graphmode,size,page;
char s[30];
graphdriver=DETECT;
initgraph(&graphdriver,&graphmode,"");
cleardevice();
setbkcolor(BLUE);
setviewport(40,40,600,440,1); /* 开图视口 */
setfillstyle(1,2);
setcolor(YELLOW);
```

```
rectangle(0,0,560,400);
floodfill(50,50,14); /* 用绿色填画出的矩形框 */
rectangle(20,20,540,380);
setfillstyle(1,13);
floodfill(2l,300,14); /* 用淡洋红色填画出的矩形框 */
setcolor(BLACK);
settextstyle(1,0,6); /* 设要显示字符串的字形方向,尺寸 */
outtextxy(100,60,"Welcom Your");
setviewport(100,200,540,380,0); /* 又开一窗口 */
setcolor(14);
setfillstyle(1,12);
rectangle(20,20,420,120);
settextstyle(2,0,9);
floodfill(21,100,14); /* 用深蓝色填充 */
sprintf(s,"Let's study Turbo C"); /* 将字符串存到 s 字符数组内 */
setcolor(YELLOW);
outtextxy(60,40, s); /* 用黄色显示 */
setcolor(1);
settextstyle(4,0,3); /* 设选用字形 4,放大 3 倍,水平设置 */
outtextxy(110,80,s); /* 显示 s 字符串 */
getch();
closegraph();
}
```

2.1.9 生成不需 Turbo C 环境支持的图形程序

1) BGI 使用

由于图形程序运行并显示图象直接与显示器有关，而如何控制驱动显示器进行显示，Tubro C 并没有向用户提供这种技术，而这也是不必要的，因它与显示器硬件结构息息相关，编程者并不需要知道这些东西，否则太复杂了。用户的图形程序要能运行并显示，则必须要包含有驱动显示器的这种程序。不同种类的显示器因硬件结构不同，因而驱动程序也不同，这些驱动程序一般由生产厂家提供，对一般标准的显示器，如 CGA、EGA，VGA 等，其驱动程序已经在 Turbo C 系统盘上提供。在用户的图形程序中，进行图形系统初始化时，即执行函数：

initgraph(&graphdriver，&graphmode，char *path_for_driver)；



时，程序就按照 path_for_driver 所指的路径将图形驱动程序装入内存。这样，以后的图形功能才能被支持。若在所指路径下找不到相应显示器的驱动程序，或没有对驱动程序进行装入操作，则运行图形程序时，就会在屏上显示出错信息：

BGI Error：Graphics not initialized(use “initgraph”)

若 Turbo C 已安装在硬盘上，则各种显示器的图形驱动程序：EGAVGA.BGI，CGA.BGI，HERC.BGI，IBM 8514.BGI，PC3270.BGI 和 BGIOBJ.EXE 等程序已展开在硬盘上，故而重复直接将其拷贝到自己的工作路径下即可，这样在集成环境下，生成的.EXE 文件，就可以在 DOS 下直接运行了。

生成能在 DOS 下直接运行的图形程序的另一种方法是：用 BGIOBJ.EXE 程序将相应显示器的驱动程序(如 CGA.BGI，EGAVGA.BGI 等)转换成.OBJ 文件，即在 DOS 下用命令(对 EGAVGA 显示器)：

C>BGIOBJ EGAVGA

这样就在用户的盘上生成了 EGAVGA.OBJ 文件，然后将该文件连到用户的库盘上 LIB 子目录下的 GRAPHICS．LIB 库中，即用 Turbo C 系统的 TLIB．EXE 文件，将 EGAVGA．OBJ

和 GRPHICS.LIB 进行连接，可用命令

C>TLIB LIB\GRAPHICS.LIB+EGAVGA.OBJ

这样就在用户的盘上的 GRAPHICS.LIB 中连入了 EGAVGA 驱动程序。

通过以上处理，无论是安装在硬盘上的 Turbo C 或者软盘上的 Turbo C，均在图形库中连入了相应显示器的驱动程序。

最后，在用户程序中的 initgraph()函数调用前要用 registerbgidriver()进行登记，表示相应的驱动程序在用户程序进行连接时已连接，因而执行 initgraph()时，不必再装入相应的 BGI 程序了，即用(对 EGA，VGA 显示器)：

registerbgidriver(EGAVGA_driver);

后，执行 initgraph()时，就不用再按该函数的第三个参数 patchfor driver 去寻找驱动程序了。

对所有的 Turbo C 图形程序，在 initgraph()函数前，加上 registerbgidriver 函数后(当然对 GRAPHICS.LIB 还要进行如上述的连接操作)，编译连接后生成的 EXE 文件，就变成了一个可独立运行的执行程序了，它不需 Turbo C 环境的支持，可在 DOS 下直接运行。

例程 3-25：下面程序假设已作了前两个步骤（在 C:\TC 子目录下输入命令:BGIOBJ EGAVGA、在 C:\TC 子目录下输入命令:TLIB LIB\GRAPHICS.LIB+EGAVGA），在 initgraph()函数调用之前加入 registerbgidriver()函数，这样就建立了一个不需要驱动程序就能独立运行的可执行图形程序。

```
#include<stdio.h>
#include<graphics.h>
int main()
{
int gdriver=DETECT,gmode;
registerbgidriver(EGAVGA_driver); / *建立独立图形运行程序 */
initgraph( gdriver,gmode,"c:\\tc");
bar3d(50,50,250,150,20,1);
getch();
closegraph();
51
return 0;
}
```

registerbgidriver(*driver(void))中的参数名和相应的图形驱动程序(.BGI)对应关系如下：

图形驱动程序 registerbgidriver 驱动器 driver 参数

CGA．BGI CGA_driver

EGAVGA．BGI EGAVGA_drvier

HERC．BGI HERC_driver

ATT．BGI ATT_driver

PC3270．BGI PC3270_driver

IBM8514．BGI IBM8514_driver

2) BGI 使用图形方式下字型输出的条件

当用文本字型定义函数

settextstyle(int font，int direction，int char size)；

设置文本输出字形时，由 font 参数给出的字型库必须装入内存，这些字型是 TRIPLEX_FONT、SMALL_FONT，SANS_SERIF_FONT 和 GOTHIC_FONT，它们对应的字型库是 trip.chr、litt.chr、sans.chr 和 goth.chr。当在程序中选中了某字形时，便将相应字库调入内存。当我们要使用这些字体输出的图形程序，在脱离 Turbo C 集成环境下，仍能独立运行，则必须将相应的字库存在图形程序盘上，其作法如同显示驱动程序一样。即将需要的程序如 trip.chr 等拷贝到程序盘上。另外也可以用 BGIOBJ 程序将相应的字库驱动程序转换成 OBJ 文件。

可用命令：BGIOBJ TRIP

这样 trip.chr 就转换成 TRIP.0BJ 了。然后用 TLIB 程序将其连接到 GRAPHICS．LIB 中，

如使用 TRLPLEX_FONT 字形时，可用命令：

C:\TC>TLIB LIB\GRAPHICS.LIB + TRIP.OBJ

这样，GRAPHICS.LIB 中就连接上了相应的字形驱动程序。

当完成上述步骤后， 还要在图形初始化之前， 即 initgraph() 前用函数

registerbgifont(void(*font)(void))进行说明，说明相应 font 的字形库已经连到 CRAPHICS．LIB

中，即连到了图形程序中。如若已装入 TRIP 库，则可写作 registerbgifont(triplex_font)。

上述的这些作法和装显示驱动程序完全一样。其中 registerbgifont()函数中的字形参数和

相应的字形驱动程序(.chr)的对应关系如下：

字形驱动程序 registerbgifont()参数名

trip.chr TRIPLEX_FONT

litt.chr SMALL_FONT

sans.chr SANSSERIF_FONT

goth.chr GOTHIC_FONT

若不经过上述过程，则含有笔划字体的图形函数独立运行时，显示的字体都以 8×8 点

阵的缺省字符显示，原来设置使用的字体将不会出现。

3) BGI 图形驱动

BGI 是 BORLAND GRAPHICS INTERFACE(BORLAND 图形接口)的简称，它是

BORLAND 公司系列编译语言如 TURBO C/C++，BORLAND C++、TURBO PASCAL，

TURBO PROLOG 图形子系统的基础，它是一种快速、紧凑型和独立于设备的软件包，其设



备独立性是通过一个公用内核(COMMON KERNEL)中调入一个可装入(Loadable)的特定设

备驱动程序来实现的，这个特定的驱动程序就是 BGI，它直接跟硬件打交道，完成图形处理

任务。显然，针对不同的硬件会有不同的内容。

目前，BORLAND 公司所提供的 6 种图形驱动 BGI 为：ATT.BGI、CGA.BGI、

EGAVGA.BGI、HERC.BGI、IBM8514.BGI、PC3270.BGI。BORLAND 系列编译语言所提供

的 BGl 支持大部分的显示卡，提供了非常丰富的图形库函数(过程)，包括画点、线、矩形、

圆、弧、二维、三维棒图、饼图、多种字体显示，各种模式填充等等，对于用户开发图形应

用程序非常方便、高效。

VGA 图形控制器是用于 IBM PC 系列计算机的最受欢迎的彩色图形系统，目前在使用

的 VGA 几乎都大大超过了 VGA 标准。这些显示卡大都提供了 640×480、800×600、1024

×768 等 16 色图形模式以及 320×200、640×350 或 640×400、800×600、1024×768 等 256

色图形模式。256 色模式中较高的分辨率已能把一整套新的成像技术用于 VGA 领域。本节

以 Trident TVGA 为例，介绍图形驱动软件包的使用。

TVGA BGI 3.0 可以购买或者在网上下载。包括 TVGAl6 BGI3.0 和 TVGA256 BGl 3.0，

它们分别支持 TVGA 卡的所有 16 色模式和所有 256 色模式，包括上述 16 色和 256 色模式

以及标准 VGA 的 16 色、256 色模式。

要使用 TVGA BGI，只要增加一下语句：

gdriver=installuserdriver(“tvga256”，detectTVGA256)；

initgraph(&gdriver,&gmode,””);

其中 gmode 为 tvga 提供的模式，可以从头文件查询到。

其他 BGI，如 SVGA.BGI，使用方法相似。

2.2 鼠标的使用

2.2.1 鼠标器简介

鼠标器有一、二或三个按钮，现普遍使用三个按钮的。目前主要有两种形式的鼠标器，

第一种是机械式的，第二种是光电式的。在机械式鼠标器中使用一个转动球，随着鼠标器移

动，转动球在旋转，它的移动使得传感器将这些移动变成移动光标的方向信息。光电式鼠标

器使用二个发光二极管 LED 和两个光电晶体管来检测移动。一个 LED 发红光，而另一个则产生紫外光，光电鼠标器用一个特殊的焊盘来改变 LED 的光强。这个焊盘有两个方向线，当鼠标器向一个方向移动时，吸收红光，向另一方向移动时，吸收紫外光，光间断的颜色和数目决定了鼠标器移动的方向和距离，它将这些信息变成数字信号向计算机发送。

鼠标器通过 RS232 异步串行口向计算机发送这些移动和移动方向及距离多少的信息。

对于 PC 机，它常接在 COMl 口或 COM2 口上，使用时主机用硬件中断 INT 0BH 接收串行口送来的数据信息(对 COM2 口)，若使用 COMl 口，则使用 INT 0CH 中断来接收 COMl 传来的信息，如同用键盘中断 INT 9 接收键盘信息一样。所以从实质上说，鼠标器如同键盘一样，是向计算机送信息的一个输入设备，它通常以 1200 波特率的传送速度和数据传送格式为 8 个数据位的方式向主机发送数据，这些数据代表着光标的移动和按钮的状态，一般定义 5 个字节为一组数据，各字节含义如下：

第一字节：80H~87H 是鼠标的三个按钮按下或松开的 8 种组合值，其中单独按下左、中、右各钮的值分别为 83H、85H、86H，三个按钮不按时，其值为 87H。

第二字节：0 没有上下方向移动，l 十表示以 l 为起点递增，数值越大，表示向右移动速度愈快。255 一，表示向左移动，即以 255 为起点向左移动，移动速度愈快，则数值递减越快。

第三字节：0 表示没有左右方向的移动，l 十表示向上移动，增加越快，表示移动越快，



255 一表示向下移动，移动越快，数值递减越快。

第四和第五字节含义与第二第三字节相似。

鼠标在移动时，其光标并没有直接显示在屏幕上，而是在一个假想的虚拟屏幕上。它以象素为单位进行计数，然后再将其映射到显示屏幕上。由于显示模式不同，即分辨率不同，满屏的象素个数不同，因而单位长度上的象素个数也不同，即鼠标计数的个数也不同。这些映射过程，都是通过鼠标器的驱动程序来完成。

一般鼠标驱动程序首先将 RS232 进行初始化，然后来用中断方式接收鼠标数据，规定采用 INT 33H 中断，即中断地址为 0000：30H~33H。鼠标驱动程序由生产鼠标的厂家提供，该程序提供了许多功能，通过设置不同的入口参数，通过 INT 33H 鼠标中断调用，来使用这些功能，如 Microsoft 的 INT 33H 中断调用，提供了 35 个功能调用。

DOS 操作系统和 Turbo C 2.0 并不支持鼠标器的操作，因而要使用鼠标器，必须首先要安装其相应的驱动程序，通常使用的方法是在 CONFIG.SYS 文件中加入一行信息：DEVICE=MOUSE.SYS，使得在 DOS 启动时，将 Mouse 的驱动程序也装入内存，或者也可直接运行 mouse.com 文件，使其驻留在内存。

2.2.2 鼠标器的 INT 33H 功能调用

当安装好了鼠标器的驱动程序，并进行了初始化后，就可使用鼠标器的驱动程序来管理鼠标器的各种操作。鼠标器驱动程序将 INT 33H 中断作为鼠标器的操作中断，这样每当移动一下鼠标器，或者按动一下鼠标器的按钮，就将产生一次 INT 33H 中断。而鼠标驱动程序将按照中断时的入口参数，调用不同的功能处理程序，来完成中断服务。

对于 Microsoft 鼠标驱动程序，共提供了 30 多个功能调用，用户可以通过 INT 33H 中断调用，选用不同的入口参数，来实现相应的功能调用，这些功能号和对应的功能如表 3-13：

表 3-13 INT 33H 中断调用功能号和对应的功能

| 功能号 | 功能简介 | 功能号 | 功能简介 |
|---|---|---|---|
| 0 | 鼠标复位及取状态 | 20 | 交换中断程序 |
| 1 | 显示鼠标光标 | 21 | 取驱动程序存储要求 |
| 2 | 鼠标光标不显示 | 22 | 保存驱动程序状态 |
| 3 | 取按钮状态和鼠标位置 | 23 | 恢复驱动程序状态 |
| 4 | 设置鼠标光标位置 | 24 | 设辅助程序掩码和地址 |
| 5 | 取按钮压下状惑 | 25 | 取用户程序地址 |

6 取按钮松开状态 26 设置分辨率

7 设置水平位置最大值 27 取分辨率

8 设置垂直位置最大值 28 设置中断速度

9 设置图形光标 29 设置显示器显示的页号

10 设置文本光标 30 取显示器显示的页号

11 取鼠标器移动的方向和距离 31 关闭驱动程序

12 设中断程序掩码和地址 32 打开驱动程序

13 打开光笔模拟 33 软件重置

14 关闭光笔模拟 34 选择语言

15 设置鼠标移动速度 35 取语言编号

16 条件关闭 36 取版本号及鼠标类型和中断号

19 设置速度加倍的下限 37 取鼠标驱动有关信息

表 3-14 列出了一些上述功能号中的基本的功能调用和相应的入口参数，及调用后的出



口参数，这些调用是一般常用的：

表 3-14 INT 33H 中断调用常用功能调用和相应的入口参数，及调用后的出口参数

功能码 功能

入口参数

出口参数

0 鼠标复位及取状态 m=0

(AX=0)

m1=-1 鼠标安装成功

(AX=-1)

m1=0 鼠标安装失败

(AX=0)

m2= 鼠标按钮数目

(BX= )

1 显示鼠标光标 m1=1

(AX=1)

无

2 不显示鼠标光标 m1=2

(AX=2)

无

3 取按钮状态和鼠标位置 m1=3

(AX=3)

m2=各按钮状态

BX= 见注释

4 设置鼠标光标位置 m1=4

(AX=4)

m3=光标 x 坐标

(CX= )

m4=光标 y 坐标

(DX= )

无

5 取按钮压下状态 m1=5

(AX=5)

m2=按钮号

(BX= )

0 为左按钮

1 为右按钮

m1=各按钮状态

(AX= 见注释)

m2=自上次调用以来该按钮

按下的次数

(BX= )

最后一次按下时鼠标的 x 坐

标

(CX= )

最后一次按下时鼠标的 y 坐

标

(DX= )

6 取按钮松开状态 m1=6

(AX=6)

m2=按钮号

(BX= )

m1=各按钮状态

(AX= 见注释)

m2=自上次调用以来该按钮

释放的次数

(BX= )

最后一次释放时鼠标的 x 坐

标

(CX= )

最后一次释放时鼠标的 y 坐

标

(DX= )

55

7 设置水平位置最大值 m1=7

(AX=7)

m3=x 坐标最小

值

m4=x 坐标最大

值

无

8 设置垂直位置最大值 m1=7

(AX=7)

m3=x 坐标最小

值

m4=x 坐标最大

值

无

11 取鼠标器移动的方向和距离 m1=11

(AX=11)

m3=x 方向移动距离

m4=y 方向移动距离

12 设中断程序掩码和地址 m1=12

m3=调用掩码，

见注释

m4=程序地址

无

注释 1：鼠标各按钮的状态，由下面信息格式提供：

位 等于 0 时 等于 1 时

0 左按钮未按下 左按钮正在按下

1 右按钮未按下 右按钮正在按下

2 中按钮未按下 中按钮正在按下

注释 2：当用功能 12 设置用户的鼠标中断服务程序时，其入口参数 m3=调用掩码，该掩码表示在哪种条件发生时，产生中断，即执行用户定义的中断服务程序。其掩码位与中断产生条件如下：(该掩码位置 1 时，便产生中断)

掩码位 中断的条件

0 光标位置移动

1 左按钮按下

2 左按钮松开

3 有按钮按下

4 右按钮松开

在上表中，各入口参数和出口参数中，m1，m2，m3，m4 分别存放在 Ax，Bx，Cx，Dx 各寄存器中。表中用括号标示出了。当用汇编语言调用上表中的功能时，只要将相应入口参数赋给相应的 Ax，Bx，Cx，Dx 寄存器即可，出口参数则从上述的寄存器中得到。

2.2.3 鼠标主要功能函数

在图形用户接口程序设计中，鼠标器光标的显示与关闭是经常要用到的功能。当进行菜单项选取或窗口移动等操作中，若不先关闭鼠标器，则有可能在鼠标器光标处留下缺憾。因此，在作图、重显等操作时，暂时关闭鼠标器光标是必要的，等作固动作完成后再重新打开光标，以保证图形界面的完美。以下例程实现光标的开关：

```
void mscurson(void) /* 显示鼠标器光标 */
{
union REGS r;
56
struct SREGS s;
r.x.ax=1; /* 1 号鼠标器功能 */
msvisible=TRUE;
int86x(0x33，&r，&r，&s);
}
void mscursoff(void) /* 关闭鼠标器光标 */
{
union REGS r;
struct SREGS s;
r.x.ax=2; /* 2 号鼠标器功能 */
msvisible=FALSE;
int86x(0x33，&r，&r，&s);
}
```

调用 4 号功能可实现鼠标器光标位置设置，该功能用于鼠标器应用程序中设定光标的初始位置。

```
void curstoxy(unsigned int x, unsigned int y)
{
union REGS r;
```

```
struct SREG s;
r.x.ax=4; /* 4 号鼠标器功能 */
r.x.cx=x;
r.x.dx=y;
int86x(0x33，&r，&r，&s);
mousex=x;
mousey=y;
}
```

初始图形状态下，驱动程序已经设定好了一组阵列来显示光标，但在应用中，可能需要根据光标的不同定位或不同功能甚至编程者的兴趣来修改光标形状。比如手型光标、双箭头光标、十字线、块状、箭头等，这一功能可用第 9 号功能实现。与文本状态下使用的长方形鼠标器光标不同，图形状态下的系统设置的形状是一个指向左上的箭头。产生用户自己的光标基本上有两种方法：一是通过调用鼠标器驱动程序的一个特殊函数；二是通过涂改产生自己的光标。

在大多数图形状态下鼠标器光标都被定义为 16×16 象素的块。虽然可通过正确设置其形状而使之变小一些，但光标的大小不能超过它。为改变鼠标器光标，可以以功能号 9 调用鼠标器驱动程序，并传三个参数给它：前两个参数定义光标的"热点"坐标。热点确定光标的原点，即从该点确定鼠标器坐标。例如，在系统设置的光标中，热点是箭头的尖。位置是相对于 16×16 块的左上角给出的，且两个参数都必须在-16 到 16 的范围内。使用负坐标，你可以把热点定义在实际光标的左上方。最后一个参数指定两个 16×16 位的用于创建光标形状的屏蔽的地址：屏幕屏蔽和光标屏蔽。当鼠标器光标画好后，屏幕屏蔽象素与屏幕上的象素进行 AND 运算。结果再与光标屏蔽进行 XOR 运算。这种有趣的组合可使你把光标制成背景色白色、透明的、或是背景色的相反色。适当设置屏蔽，你可以在同一光标内得到这些颜色的任一组合。表 3-15 表明位组合如何起作用。

表 3-15 鼠标位组合方法

57

屏幕屏蔽 光标屏蔽 结果颜色

0 0 背景色

0 1 白色

1 0

透明

1 1 前景反转，背景白色

屏幕屏蔽阵列与光标屏蔽阵列的表示法可使用具有 32 个元素的 unsisned int 阵列，前 16 元素是屏幕屏蔽，后 16 个元素代表光标屏蔽，而每一个元素各代表同一列上的 16 个点(因为一个 unsingned int 具有 16 个位)，所以整个阵列代表一个 16×32 的点象素阵列。

例程 3-26：下面阵列的前半部是屏幕屏蔽，后半部是光标屏蔽。可看出光标的样式为一个指向左上角的箭号，其标示点可定义为(0，0)以代表左上角的点。

```
unsigned int pattern[32]= {
/* 屏幕掩码 */
0x3FFF, /* 0011111111111111 */
0x1FFF, /* 0001111111111111 */
0x0FFF, /* 0000111111111111 */
0x07FF, /* 0000011111111111 */
0x03FF, /* 0000001111111111 */
0x01FF, /* 0000000111111111 */
0x00FF, /* 0000000011111111 */
0x007F, /* 0000000001111111 */
0x003F, /* 0000000000111111 */
```

```
0x001F, /* 0000000000011111 */
0x01FF, /* 0000000111111111 */
0x10FF, /* 0001000011111111 */
0x30FF, /* 0011000011111111 */
0xF87F, /* 1111100001111111 */
0xF87F, /* 1111100001111111 */
0xFC3F, /* 1111110000111111 */
/* 光标屏蔽 */
0x0000, /* 0000000000000000 */
0x4000, /* 0100000000000000 */
0x6000, /* 0110000000000000 */
0x7000, /* 0111000000000000 */
0x7800, /* 0111100000000000 */
0x7C00, /* 0111110000000000 */
0x7E00, /* 0111111000000000 */
0x7F00, /* 0111111100000000 */
0x7F80, /* 0111111110000000 */
0x7FC0, /* 0111111111000000 */
0x6C00, /* 0110110000000000 */
0x4600, /* 0100011000000000 */
0x0600, /* 0000011000000000 */
58
0x0300, /* 0000001100000000 */
0x0300, /* 0000001100000000 */
0x0180 /* 0000000110000000 */
};
```

以下函数进行鼠标光标设置。

```
void set_graphic_sursor(int x, int y, unsigned int far *pattern)
{
union REGS ireg;
struct SREG isreg;
ireg.x.ax=9; /*/* 9 号鼠标器功能 */
ireg.x.bx=x;
ireg.x.cx=y;
ireg.x.dx=FP_OFF(pattern);
isreg.es=FP_SEG(pattern);
int86x(0x33, &ireg, &ireg, &isreg);
}
```

传入值：x 及 y 为"热点"的坐标，而 patterm 则为屏蔽阵列的指针。

鼠标器状态查询是鼠标器应用程序中应用最为颠繁的函数，由第 3 号功能实现。该函数报告鼠标器当前光标位置和当前按键状态。例程如下：

```
void mouseread() {
union REGS rl,r2;
struct SREGS s;
r1.x.ax=3; /* 3 号鼠标器功能 */
int86x(0x33，&r1，＆r2，&s)；
mousex=r2.x.cx; /* 鼠标器光标 x 方向坐标 */
mousey=r2.x.dx; /* 鼠标器光标 y 方向坐标 */
```

mousekey=r2.x.bx; /* 鼠标器按键状态 */

}

其中的 mousex、mousey 为全局变量，记录鼠标器光标位置。mousekey 也是一个外部变量，记录鼠标器按键动作。

在某些情况下，需要等待鼠标器某种状态，比如按键按下。下面的程序提供这种功能，它等待直到按键按下或移动光标才返回调用程序。

```
void wait(int i)
{
do {
mouseread();
} while(mousekey==i);
}
```

以上六种功能是最为常用的，其它功能可类似实现。

2.3 问题实现



本节通过调用 INT 33H 软中断的不同功能实现最初所提问题(如图 3-16 所示，例程 3-27)。

最后的程序演示了如何调用这些功能，为了使程序便于理解，将作图时要用到的一些功能调用以函数形式实现，这些函数有：

init(int xml，int xma，int ymi，int yma)；

该函数将通过调用 int 33H 的 0 号功能调用对鼠标器进行初始化，调用 7 号和 8 号功能，设置 x 和 y 位置的最小和最大值。这就为鼠标器移动进行了初始化准备。由于 0 号功能调用是测试鼠标驱动程序是否安装，因此在运行该程序前必须首先执行鼠标驱动程序 mouse.com，若调用该函数执行了 0 号功能调用，当返回值为 0 时(即返回参数为 0)，表示未安装成功，这可能是鼠标器或驱动程序未安装。这时程序将显示 Mouse or Mouse Driver Absent，并回到系统。

read(int *mx，int *my，int *mbutt)

该函数将通过调用 int 33H 的 3 号功能调用，读鼠标的位置和按钮状态。鼠标的 x、y 位置值将由指针 mx 和 my 给出，而按钮状态则由 mbutt 指针给出。

cursor(int x，int y)；

该函数将用画线函数 line()画出一个十字形光标。

newxy(int *mx，int *my，int *mbutt)；

该函数将通过调用 read()函数来判断是否有按钮按下，若按下，则调用 cursor()函数在新位置画出一十字光标。

光标的移动是通过将原位置光标用背景色再画而使其消失，然后在新位置处重新画一个光标，从而实现光标移动的动感。

当光标移动到 quit 上时，程序便终止，并回到系统。

运行该程序前，要将鼠标器安装在 COM1 口，并在当前目录下运行 mouse 驱动程序，然后运行该程序，按住鼠标器的任意键并移动，十字光标将随鼠标而移动。当仅按下左键时，用圆圈在蓝色背景上画出移动的轨迹，仅按下右键时，用矩形，其它按键情况用线条。

```
/*例程 3-27*/
#include <dos.h>
```

图 3-16 用鼠标进行作图



```
#include <stdio.h>
#include <graphics.h>
union REGS regs;
int init();
```

```
int read();
void cursor(),newxy();
int xmin,xmax,ymin,ymax,x_max=639,y_max=479;
main()
{
int buttons,xm,ym,x0,y0,x,y;
char str[100];
int driver=VGA;
int mode=VGAHI;
initgraph(&driver,&mode,"");
clrscr();
rectangle(0,0,x_max,y_max);
setfillstyle(SOLID_FILL,BLUE);
bar(1,1,x_max-1,y_max-1);
outtextxy(3,15,"move mouse using any button.");
outtextxy(285,15,"quit");
xmin=2;
xmax=x_max-1;
ymin=8;
ymax=y_max-2;
setwritemode(XOR_PUT);
if(init(xmin,xmax,ymin,ymax)==0 ) /* 调用 init 函数对鼠标器初始化*/
{
printf("Mouse or Mouse Driver Absent,P1ease install!");
delay(5000);
exit(1);
}
x=320;y=240;
cursor(x,y); /* 置十字光标在屏幕中心。 */
for(;;) {
newxy(&x,&y,&buttons);
if(x>=280&& x<=330 &&y>=12&&y<=33&& buttons) /*十字光标移到 quit 处时
*/
{
cleardevice();
exit(0); /* 回到系统*/
}
}
}
void cursor(int x,int y) /* 画十字光标函数 */
61
{
int x1,x2,y1,y2;
x1=x-4;
x2=x+4;
y1=y-3;
y2=y+3;
line(x1,y,x2,y);
```

```
line(x,y1,x,y2);
}
int init(int xmi,int xma,int ymi,int yma) /* 鼠标器初始化函数 */
{
int retcode;
regs.x.ax=0;
int86(51,&regs,&regs);
retcode=regs.x.ax;
if(retcode==0)
return 0; /* 返回 0 表示鼠标或鼠标驱动程序未安装 */
regs.x.ax=7;
regs.x.cx=xmi;
regs.x.dx=xma;
int86(51,&regs,&regs);
regs.x.ax=8;
regs.x.cx=ymi;
regs.x.dx=yma;
int86(51,&regs,&regs); /* 表示鼠标器和驱动程序已安装 */
return retcode;
}
int read(int *mx,int *my,int *mbutt) /* 读鼠标的位置和按钮状态函数 */
{
int xx0=*mx,yy0=*my,but0=0,mb;
int xnew, ynew;
do {
regs.x.ax=3;
int86(51,&regs,&regs);
xnew=regs.x.cx;
ynew=regs.x.dx;
*mbutt=regs.x.bx;
} while(xnew==xx0 && ynew==yy0 && *mbutt == but0 );
*mx=xnew;
*my=ynew;
mb=(*mbutt);
if(mb){
if(mb==1) return 1; /*左键按下*/
if (mb==2) return 2; /*右键按下*/
62
return 3; /*其它的按键情况*/
}
else
return 0;
}
void newxy(int *mx,int *my,int *mbutt) /* 是否有按钮按下，若按下，在新位置画十
字 */
{
int ch,xx0=*mx,yy0=*my,x,y;
int xm,ym;
```

```
ch=read(&xm,&ym,mbutt);
switch (ch) {
case 0:
cursor(xx0,yy0);
cursor(xm,ym);
break;
case 1:
cursor(xx0,yy0);
cursor(xm,ym);
circle(xm,ym,6);
break;
case 2:
cursor(xx0,yy0);
cursor(xm,ym);
rectangle(xm,ym,xm+12,ym+12);
break;
case 3:
cursor(xx0,yy0);
cursor(xm,ym);
putpixel(xm,ym,7);
break;
}
*mx=xm;
*my=ym;
}
```

实验二

(1) 当给定一年内各月国民生产总值，绘制一个立体柱状图表现它（如图 3-17 所示）。屏幕 x 轴为月份，y 轴为产值，y 最大值设置为屏幕最高，y 起点为 0。柱状图的前面为白色，顶面的平行四边形为蓝色，侧面使用粗"//" 填充。 图 3-17 年国民生产总值的月立体柱状图



(2) 编写程序，使用鼠标进行如下操作：鼠标未按键时可以移动，鼠标按着左键移动时为画矩形，鼠标按着右键移动为画椭圆，鼠标左键双击为使用红色填充当前点所在的连通域，鼠标右键双击为退出程序。

(3) 改进第 2 章的扫雷游戏，完成图形方式下的菜单实现及雷区绘制，界面如图 3-5 所示。

(4) 修改上一步改进后的扫雷游戏，将其中的键盘操作方式改为鼠标方式。

3 屏幕图象与动画技术

[问题的提出] 编写程序，该程序将在繁星点缀的黑色背景中显示一个经纬线为蓝色的并围绕着一红色光环的地球，光环时隐时现，地球也在自西向东转动，一蓝色宇宙飞船从左自右缓缓飞过，周而复始。屏幕下方显示 AROUND THE WORLD 字样。

[分析] 在上面的问题中，我们已能解决的是在图形模式下画静态的图形，例如产生星空背景。当然，若地球、光环甚至宇宙飞船不是动态的，我们同样可以完成。一旦一个图形画在了屏幕上就成为整个屏幕图象的一部分。那么如何产生动态的画面呢？我们知道电影或动画

片是由一张张图象组成的，利用人眼不能够分辨出时间间隔在 25 毫秒内的动态图象变化这一特性，当这些连续图象被放映时，从视觉效果上给人以动的感觉。所以在计算机屏幕上产生运动的效果需要动画技术。

[解答] 下面我们将介绍几种用 Turbo C 实现动画的方法，这将涉及到一些我们在 3.2 节中没有讲述的图形处理函数。

3.1 一个简单的实现方法

这种简单方法利用 cleardevice()和 delay()函数相互配合，先画一幅图形，让它延迟一个时间，然后清屏，再画另一幅，如此反复，形成动态效果。本小节的例程 3-28 分别通过函数 graphone()，graphtwo()和 graphthree()实现了三个简单的动画画面，这三个画面不停地进行切换。

```
/*例程 3-28*/
#include<graphics.h>
#include<stdlib.h>
#include<dos.h>
int x,y,maxcolor;
void graphone(char *str); /*这个函数实现的是字符串 str 左右运动，线条上下运动*/
void graphtwo(char *str); /*这个函数实现的是字符串 str 上下运动，线条左右运动*/
void graphthree(char *str);
/*这个函数实现的是字符串 str 由小变大，再由大变小，直线也随之变化*/
main()
{
int i,driver,mode;
char *str="W E L C O M E !";
driver=DETECT;
mode=0;
initgraph(&driver,&mode,""); //系统初始化
cleardevice(); // 清屏
settextjustify(CENTER_TEXT,CENTER_TEXT); // 设置字符串的定位模式
x=getmaxx(); // 返回当前图形模式下的最大有效的 x 值
64
y=getmaxy(); // 返回当前图形模式下的最大有效的 y 值
maxcolor=getmaxcolor(); // 返回当前图形模式下的最大有效的颜色值
while(!kbhit())
{
graphone(str); // 第一个动画
graphtwo(str); // 第二个动画
graphthree(str); // 第三个动画
}
getch();
closegraph(); // 关闭图形模式
}
void graphone(char *str)
{
int i;
for(i=0;i<40;i++)
{
setcolor(1);
settextstyle(1,0,4);
```

```
setlinestyle(0,0,3);
cleardevice();
line(150,y-i*15,150,y-300-i*15);
line(170,y-i*15-50,170,y-350-i*15);
line(130,y-i*15-50,170,y-i*15-50);
line(150,y-300-i*15,190,y-300-i*15);
line(x-150,i*15,x-150,300+i*15);
line(x-170,i*15-50,x-170,250+i*15);
line(x-150,i*15,x-190,i*15);
line(x-130,250+i*15,x-170,250+i*15);
outtextxy(i*25,150,str);
outtextxy(x-i*25,y-150,str);
delay(5000);
}
}
void graphtwo(char *str)
{
int i;
for(i=0;i<30;i++)
{
setcolor(5);
cleardevice();
settextstyle(1,1,4);
65
line(i*25,y-100,300+i*25,y-100);
line(i*25,y-120,300+i*25,y-120);
line(x-i*25,100,x-300-i*25,100);
line(x-i*25,120,x-300-i*25,120);
outtextxy(150,i*25,str);
outtextxy(x-150,y-i*25,str);
delay(5000);
}
}
void graphthree(char *str)
{
int i,j,color,width;
color=random(maxcolor); // 随机得到颜色值
setcolor(color);
settextstyle(1,0,1); // 设置字符串的格式
outtextxy(x/2,y/2-100,str); // 显示字符串
delay(8000);
for(i=0;i<8;i++) // 字符串由小变大
{
cleardevice(); // 清屏
settextstyle(1,0,i);
outtextxy(x/2,y/2-i*10-100,str);
outtextxy(x/2,y/2+i*10-100,str);
width=textwidth(str); // 得到当前字符串宽度
```

```
setlinestyle(0,0,1); // 设置画线格式
line((x-width)/2+10*(8-i),y/2+i*15-70,(x+width)/2-10*(8-i),y/2+i*15-70);
line((x-width)/2+5*(8-i),y/2+i*15-60,(x+width)/2-5*(8-i),y/2+i*15-60);
line((x-width)/2,y/2+i*15-50,(x+width)/2,y/2+i*15-50);
line((x-width)/2,y/2+i*15-20,(x+width)/2,y/2+i*15-20);
line((x-width)/2+5*(8-i)-10,y/2+i*15-10,(x+width)/2-5*(8-i),y/2+i*15-10);
line((x-width)/2+10*(8-i),y/2+i*15,(x+width)/2-10*(8-i),y/2+i*15);
delay(8000);
}
for(i=7;i>=0;i--) // 字符串由大变小
{
cleardevice(); // 清屏
settextstyle(1,0,i);
outtextxy(x/2,y/2-i*10-100,str);
outtextxy(x/2,y/2+i*10-100,str);
width=textwidth(str);
66
setlinestyle(0,0,1);
line((x-width)/2+10*(8-i),y/2+i*15-70,(x+width)/2-10*(8-i),y/2+i*15-70);
line((x-width)/2+5*(8-i),y/2+i*15-60,(x+width)/2-5*(8-i),y/2+i*15-60);
line((x-width)/2,y/2+i*15-50,(x+width)/2,y/2+i*15-50);
line((x-width)/2,y/2+i*15-20,(x+width)/2,y/2+i*15-20);
line((x-width)/2+5*(8-i),y/2+i*15-10,(x+width)/2-5*(8-i),y/2+i*15-10);
line((x-width)/2+10*(8-i),y/2+i*15,(x+width)/2-10*(8-i),y/2+i*15);
delay(8000);
}
}
```

程序中用到的库有 graphics.h，dos.h，和 stdlib.h，其中 graphics.h 中的图形库函数，除 initgraph()，cleardevice()，closegraph()，settextjustify()，settextstyle()，setlinestyle()，outtextxy()，setcolor()，line()外，还包括

1) getmaxx()

原型说明：void far getmaxx (void)

主要功能：返回当前图形模式下的最大有效的 x 值（即最大的横坐标）

2) getmaxy()

原型说明：void far getmaxy (void)

主要功能：返回当前图形模式下的最大有效的 y 值（即最大的纵坐标）

3) getmaxcolor()

原型说明：void far getmaxcolor(void)

主要功能：返回当前图形模式下的最大有效的颜色值

4) textwidth()

原型说明：void far textwidth(char far *str)

主要功能：以象素为单位，返回由 str 所指向的字符串宽度，针对当前字符的字体与大小。

该程序用到的 dos.h 中的库函数有 delay()，其原型说明为：

void delay(unsigned milliseconds);

该函数将程序的执行暂停一段时间(毫秒)

该程序用到的有 stdlib.h 的库函数 random()random()，其原型说明为：

void far random(int num)

此函数返回一个 0-num 范围内的随机数。

3.2 利用动态开辟图视口的方法

在 3.2.1.7 节介绍了图视口的操作函数，这样我们可以利用图视口设置技术，可以实现图视口动画效果，例如可在不同图视口中设置同样的图象，而让图视口沿 x 轴方向移动设置，这次出现前要清除上次图视口的内容，这样就会出现图象沿 x 轴移动的效果。也就是说，在位置动态变化，但大小不变的图视口中(用 setviewpot()函数)，设置固定图形(也可是微小变化的图象)，这样虽呈现在观察者面前的是当前图视口位置在动态变化，但视觉上却象是看到图象在屏幕上动态变化一样。

例程 3-29 就是这样做的，不断的沿 x 轴开辟图视窗口，就像一个大小一样的窗口沿 x 轴在移动，由于总有 clearviewport 函数清除上次窗口的相同立方体，因而视觉效果上，就像



一个立方体从左向右移动一样。程序中定义的 movebar 函数作用是开辟一个图视窗口，并画一个填色的立方体，保留一阵（delay(250000)）然后清除它，主程序不断调用它，因每次顶点 x 坐标在增加，因而效果是立方体沿 x 轴从左向右在运动。

```
/*例程 3-29*/
#include <graphics.h>
#include <dos.h>
main()
{
int i,graphdriver,graphmode;
graphdriver=DETECT;
initgraph(&graphdriver,&graphmode,"");
for(i=0;i<25;i++)
{
setfillstyle(1,i);
movebar(i * 20);
}
closegraph();
}
movebar(int xorig) /* 设窗口并画填色小立方体 */
{
setviewport(xorig,0,639,199,1);
setcolor(5);
bar3d(10,120,60,150,40,1);
floodfill(70,130,5);
floodfill(30,110,5);
delay(250000);
clearviewport();
}
```

采用上面的两种方法对较复杂图形不宜，一则画图形要占较长时间，二则图视口位置切换的时间就变得较长，因而动画效果就会变差。

3.3 利用屏幕图象存储再放的方法

在图形方式下，与文本方式类似，除了清屏函数 cleardevice()外，还有其它的对屏幕图象操作的函数，其中一类是屏幕图象存储和显示函数，包括：

1) 存屏幕图象到内存区

void far getimage(int x1，int y1，int x2，int y2，void far *bitmap)；

该函数将把屏幕左上角为(x1，y1)，右下角为(x2，y2)矩形区内的图象保存到指针 bitmap 指向的内存区去。为了能开辟一个内存缓冲区，使它恰能存下所指矩形区中的图象，则必须

首先要知道所存图象占多少字节，则内存缓冲区也可设这样多的字节，这可用下面的函数：

2）测定图象所占字节数的函数

unsigned far imagesize(int x1，int y1，int x2，int y2)；

该函数将得到屏幕上左上角为(x1，y1)，右下角为(x2，y2)矩形区内图象所占的字节数。

3) 将所存图象显示的函数

void far putimage(int x1，int y1，void far *bitmap，int op)；



该函数将把指针 bitmap 指向的内存区中所装图象，与屏上现有左上角为(xl，y1)的矩形区内图象进行 op 规定的操作（参见表 3-16）。该函数进行各种图象的逻辑操作如同二进制操作一样。

表 3-16 op 规定值及操作

符号名 值 含义

COPY_PUT 0 复制

XOR_PUT 1 进行异或操作

OR_PUT 2 进行或操作

AND_PUT 3 进行与操作

NOT_PUT 4 进行非操作

例程 3-30 演示了表 3-16 中的逻辑操作，for 循环用来在屏上方产生连续的五个方框，方框中套一用洋红色（5）填充的小方块，五个图象全一样。循环结束后，又在屏下方画出两个框，小框用洋红色填充并在大框内。程序运行后，立即在屏上显示出上述图案，当按任一键后，则由函数 imagesize 得到屏下方大框套一填充框区域内图象所占字节数，然后由 malloc 函数按字节数分配一内存缓冲区 buffer，再由 getimage 函数将图象存到 buffer 中，然后复制到屏上方左边第一个框位置。按任一键后，又将 buffer 中图象和第二个框图象进行与操作后显示，再按任一键，buffer 中的图象又和第三个方框内图象进行或操作并显示，如此重复，则可将 5 种逻辑操作结果均显示在屏上。注意 COPY 和 NOT 操作将与原来屏上的图象无关，buffer 中图象经过这两种操作，将覆盖掉原屏上图象，并将结果进行显示。

```
/*例程 3-30*/
#include <graphics.h>
main()
{
int i,j,graphdriver,graphmode,size;
void *buffer;
graphdriver=DETECT;
initgraph(&graphdriver,&graphmode,"");
setbkcolor(BLUE);
cleardevice();
setcolor(YELLOW);
setlinestyle(0,0,1); /* 用细实线 */
setfillstyle(1,5); /* 用洋红实填充 */
for(i=0;i<5;i++) /* 产生连续的五个方框中套小框的 */
{
j=i *110;
rectangle(80+j,100,130+j,150); /* 产生小框且用洋红色填充 */
floodfill(110+j,140,YELLOW);
rectangle(50+j,100,130+j,180); /* 画大框 */
}
rectangle(50,340,100,420); /* 产生一个小框 */
floodfill(80,360,YELLOW); /* 用洋红色填充 */
```

```
rectangle(50,340,130,420); /* 产生一个大框 */
getch();
```

```
size=imagesize(40,300,132,430);/* 取得(40,300)右下角(132,430)区域图
象字节数 */
buffer=malloc(size); /* 分配缓冲区(按字节数) */
getimage(40,300,132,430,buffer); /* 存图象 */
putimage(40,60,buffer,COPY_PUT); /* 重新复制 */
getch();
j=110;
putimage(40+j,60,buffer,AND_PUT); /* 和屏上的图与操作 */
getch();
putimage(40+2*j,60,buffer,OR_PUT);
getch();
putimage(40+3*j,60,buffer,XOR_PUT);
getch();
putimage(40+4*j,60,buffer,NOT_PUT);
getch();
closegraph();
}
```
同制作幻灯片一样，将整个动画过程变成一个个片断，然后存到显示缓冲区内，当把它们按顺序重放到屏幕上时，就出现了动画效果，这可以用 getimage()和 putimage()函数来实现，这种方法较快，因它已事先将要重放的画面画好了。余下的问题，就是计算应在什么位置重放的问题了。

例程 3-31 演示了利用这种方法产生的两个洋红色小球碰撞、弹回、又碰撞，……的动画效果。

```
/*例程 3-31*/
#include <graphics.h>
main()
{
int i,graphdriver,graphmode,size;
void *buffer;
graphdriver=DETECT;
initgraph(&graphdriver,&graphmode,"");
setbkcolor(BLUE);
cleardevice();
setcolor(YELLOW);
setlinestyle(0,0,1);
setfillstyle(1,5);
circle(100,200,30);
floodfill(100,200,YELLOW);/*填充圆*/
size=imagesize(69,169,131,231);/* 指定图象占字节数*/
buffer=malloc(size); /* 分配缓冲区(按字节数) */
getimage(69,169,131,231,buffer); /* 存图象 */
putimage(500,169,buffer,COPY_PUT); /* 重新复制 */
do{
for(i=0;i<185;i++)
```

```
{
putimage(70+i,170,buffer,COPY_PUT); /*左边球向右运动*/
putimage(500-i,170,buffer,COPY_PUT); /*右边球向左运动*/
}/*两球相撞后循环停止*/
for(i=0;i<185;i++)
{
putimage(255-i,170,buffer,COPY_PUT); /*左边球向左运动*/
putimage(315+i,170,buffer,COPY_PUT); /*右边球向右运动*/
}
}while (!kbhit());/*当不按键时重复上述过程*/
getch();
closegraph();
}
```

3.4 利用页交替的方法

对屏幕图象操作的函数，还有一类是设置显示页函数。我们曾在 3.2.1.1 节中提到了显示适配器的显示存储器 VRAM，图形方式下存储在 VRAM 中的一满屏图象信息称为一页。每个页一般为 64K 字节，VRAM 可以存储要显示的图象几个页(视 VRAM 容量而定，最大可达 8 页)，Turbo C 支持页的功能有限，按在图形方式下显示的模式最多支持 4 页(EGALO 显示方式)，一般为两页(注意对 CGA，仅有一页)，因存储图象的页显示时，一次只能显示一页，因此必须设定某页为当前显示的页(又称可视页)，缺省时定为 0 页，如图 3-18 所示。正在由用户编辑图形的页称为当前编辑页(又称激活的页)，这个页不等于显示页，即若用户不设定该页为当前显示页时，在该页上编辑的图形将不会在屏上显示出来。缺省时，设定 0 页为当前编辑页，即若不用下述的页设置函数进行设置，就认定 0 页既是编辑页，又是当前显示页。

设置激活页和显示页的函数如下：

```
void far setactivepage(int pagenum);
void far setvisualpage(int pagenum);
```

这两个函数只能用于 EGA、VGA 等显示适配器。前者设置由 pagenum 指出的页为激活的页，后者设置可显示的页。当设定了激活的页，即编辑页后，则程序中其后的画图操作均在该页进行，若它不定为显示页，则其上的图象信息并不会在屏上显示出来。关于图形模式和可以设定的页数，可参阅表 3-4。

例程 3-32：下面的程序演示了设置显示页函数的应用。首先用 setactivepage(1)设置 l 页为编辑页，在上面画出一个红色边框、用淡绿色填充的圆，此图并不显示出来(因缺省时，定义 0 页为可视页)。接着又定义 0 页为编辑页并清屏(即清 0 页)，也定义 0 页为可视页，并在其上画出一个用洋红色填充的方块，该方块将在屏上显示出来。接着进入 do 循环，设置 l 页为可视页，因而其上的圆便在屏上显示出来，方块的图象消失，用 delay(2000)将圆图象保

图 3-18 VRAM 中显示页与显示的关系



持 2000 毫秒即 2 秒，当不按键时，下一次循环又将 0 页设为可视页，因而方块的图象显示出来，圆图象又消失。保持 2 秒后，又重复刚开始的过程。这样我们就会看到：屏上同一位置洋红色圆和淡绿色方块交替出现，若将 delay 时间变少，将会出现动画的效果。

```
/*例程 3-32*/
#include <graphics.h>
#include <dos.h>
main()
{
int i,graphdriver,graphmode,size,page;
graphdriver=DETECT;
```

```
initgraph(&graphdriver,&graphmode,"");
cleardevice();
setactivepage(1); /* 设置 l 页为编辑页 */
setbkcolor(BLUE);
setcolor(RED);
setfillstyle(1,10);
circle(130,270,30); /* 画圆 */
floodfill(130,270,4); /* 用淡绿色填充圆 */
setactivepage(0); /* 设置 0 页为编辑页 */
cleardevice(); /* 清 0 页 */
setfillstyle(1,5);
bar(100,210,160,270); /* 画方块并填充洋红色 */
setvisualpage(0); /* 设置 0 页为可视页 */
page=1;
do
{
setvisualpage(page); /* 显示设定页的图象 */
delay(2000); /* 延迟 2000ms */
page=page-1;
if(page<0)
page=1;
} while(!kbhit());
getch();
closegraph();
}
```

正如上面的程序所示，将当前显示页和编辑页分开(用 setvisualpage()和 setactivepage()函数)，在编辑页上画好图形后，立即令该页变为显示页显示，然后在上次的显示页上(现在变为编辑页)进行画图，画好后，又再次交换，如此编辑页和显示页反复地交换，在观察者的视觉上，就出现了动画的效果。要让页的交替速度快，唯一的办法是缩短在页上的画图时间。

3.5 问题实现

在此我们主要采用了 3.3.3 介绍的方法。程序（例程 3-33）首先用 outtextx()函数在屏幕下方显示了 AROUND THE WORLD 字样，然后调用 draw_image(x,y)函数画出尾部带有三个



天线的飞船，用 imagesize()函数求出了该函数所占字节数，然后用 pt_add 指针指向存放该图形的缓冲区，并将飞船图形存在该缓冲区，接着调用 putstar()函数画星星，该函数用了初始化随机数发生器函数 srand()，和随机数发生器 random(r)，srand 使得 random()每次重新产生新起点的随机数，该随机数为 0~r-1，这样就将在画面随机地产生由小圆点和象素点构成的夜空小星星画面。

在 while 循环中，当不按键时，就反复产生一个红色光环，接着又是黑色光环，这实际上使得产生的红色光环时隐时现，因而给人以动的感觉，接着的 for 循环则用来产生地球的经纬线，它们实际上是由不同长短半径的椭圆组成，给人以立体感。为了造成动的感觉，使当 i 为偶数时为浅蓝色，奇数时为黑色，这样看起来就象地球自西向东转一样。纬线则不动。接着用 putimage()将飞船图象以每次 x 方向增加 6 复现在屏幕上，当 x 达到最大边界 max 时，便重新从 x=2 处开始。这个 while 循环中的第一个 putimage()将飞船画面与原来的画面进行异或操作，从而实现原画面的恢复工作，第二个 putimage()将在新位置让飞船出现，下一轮循环时，将由第一个 putimage 将其覆盖并恢复原屏幕图象，这样就实现了飞船的飞行。

```
/*例程 3-33*/
```

```
#include<graphics.h>
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define IMAGE_SIZE 10
void draw_image(int x,int y);
void putstar(void);
main()
{
int driver=DETECT;
int mode,color;
void *pt_addr;
int x,y,maxy,maxx,midy,midx,i;
unsigned size;
initgraph(&driver,&mode,"");
maxx=getmaxx(); /*取允许的最大 x 值*/
maxy=getmaxy(); /*取允许的最大 y 值*/
midx=maxx/2;
x=0;
midy=y=maxy/2;
settextstyle(TRIPLEX_FONT,HORIZ_DIR,4);
settextjustify(CENTER_TEXT,CENTER_TEXT);
outtextxy(midx,400,"AROUND THE WORLD");
setbkcolor(BLACK);
setcolor(RED);
setlinestyle(SOLID_LINE,0,THICK_WIDTH);
ellipse(midx,midy,130,50,160,30);
73
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
draw_image(x,y); /*画飞船*/
size=imagesize(x,y-IMAGE_SIZE,x+(4*IMAGE_SIZE),y+IMAGE_SIZE);
pt_addr=malloc(size);
getimage(x,y-IMAGE_SIZE,x+(4*IMAGE_SIZE),y+IMAGE_SIZE,pt_addr);
putstar();
setcolor(WHITE);
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
rectangle(0,0,maxx,maxy); /*画方框*/
while (!kbhit())
{
putstar();
setcolor(RED); /*画一个围绕地球的光环*/
setlinestyle(SOLID_LINE,0,THICK_WIDTH);
ellipse(midx,midy,130,50,160,30);
setcolor(BLACK);
ellipse(midx,midy,130,50,160,30);
for (i=0;i<=13;i++) /*画地球*/
{
setcolor(i%2==0 ? LIGHTBLUE:BLACK);
```

```
ellipse(midx,midy,0,360,100-8*i,100);
setcolor(LIGHTBLUE);
ellipse(midx,midy,0,360,100,100-8*i);
}
putimage(x,y-IMAGE_SIZE,pt_addr,XOR_PUT);/*恢复原画面*/
x=x>=maxx?0:x+6;
putimage(x,y-IMAGE_SIZE,pt_addr,COPY_PUT);/*在另一个位置显示飞船*/
}
free(pt_addr);
closegraph();
return;
}
void draw_image(int x,int y) /*画飞船*/
{
int arw[11];
arw[0]=x+10; arw[1]=y-10;
arw[2]=x+34; arw[3]=y-6;
arw[4]=x+34; arw[5]=y+6;
arw[6]=x+10; arw[7]=y+10;
arw[8]=x+10; arw[9]=y-10;
moveto(x+10,y-4);
setcolor(14);
74
setfillstyle(1,4);
linerel(-3*10,-2*8); /*画尾部天线*/
moveto(x+10,y+4);
linerel(-3*10,+2*8);
moveto(x+10,y);
linerel(-3*10,0);
setcolor(3);
setfillstyle(1,LIGHTBLUE);/*画本体*/
fillpoly(4,arw);
}
void putstar(void) /*画星星*/
{
int seed=1858;
int i,dotx,doty,h,w,color,maxcolor;
maxcolor=getmaxcolor(); /*得到当前模式和最多颜色数*/
w=getmaxx();
h=getmaxy();
srand(seed);
for(i=0;i<250;++i)
{
dotx=i+random(w-1);
doty=1+random(h-1);
color=random(maxcolor);
setcolor(color);
putpixel(dotx,doty,color);/*用点表示小星*/
```

```
circle(dotx+1,doty+1,1);/*用圆表示大星*/
}
srand(seed);
}
```

实验三

(1) 改造扫雷游戏，用图片来代替开局界面中的笑脸。

(2) 改造扫雷游戏，在界面中显示游戏进展的状态条（根据雷数）。

(3) 完善扫雷游戏，在开局或结束游戏时播放一段你自己设计的动画片段。

4 中断技术

[问题的提出] 在大程综合实验选题时，很多同学选择游戏。对于两人对玩的游戏来说，需要共同使用键盘，大家往往运用在 3.1.2 节介绍的知识，实现的思路往往是这样的（以单机版的对抗的俄罗斯方块为例）：

```
while(非程序退出条件)
{
for(i = 0; i < 一定时间内; ++i)
{
75
if(bioskey(1)!=0)
{
调用 bioskey（0）
if(有效按键 1) 处理;
if(有效按键 2) 处理;
…
}
自动掉下。
}
```

结果会遇到麻烦：在对玩游戏的激战中，往往会发生这样的情况，一个人按住键不放，另一个人的按键就会失效或者是一直按下键的就不能响应了。具体可见下面示例：

```
void main()
{
int i = 0, j = 0, k = 0, m = 0;
for(i = 0; i<10000; ++i)
for(j = 0; j < 10000; ++j)
{
for(k = 0; k <10000; ++k)
for(m = 0; m <10000; ++m);
if(bioskey(1)!=0)
{
if(bioskey(0)==18432)/*key_up */
printf("%d key UP pressed\n", bioskey(0));
if(bioskey(0)==8051)/*key_s */
printf("%d key S pressed\n", bioskey(0));
}
}
}
```

如何解决该问题？

[分析] 我们再来看看键盘的响应。当在键盘上按下一个键时，键盘上的处理器首先向计算机主机发出硬件中断请求，然后将该键的扫描码以串行的方式传送给计算机主机，计算机主

机在硬件中断的作用下，调用 INT 09H 硬件中断把键盘送来的扫描码读入，并转换为 ASCII 码存入键盘缓冲区中。按下一个键，送出一个闭合码，键被释放时送出一个断开码，键盘处理中断程序从键盘 I/O 端口（端口地址为 60H）读取一个字节的数据，如果读取的数据的第 7 位为 1 时表示按键已放开（送出断开码），如第 7 位为 0 表示键按下（送出闭合码），数据的第 0 - 6 位则为按键的扫描码。我们在上述问题中是通过 bioskey()使用了原有的键盘中断程序。使用 PC 机原有的键盘处理程序可以很方便地处理键盘，但是因为它是调用 BIOS，所以反应比较慢，另外当我们要同时处理几个按键时，原有的键盘中断程序就不能满足要求，这时就需要编写一个适合我们要求的键盘中断程序。

[解答]

中断技术我们在第一章就让大家开始了解，之后我们实际上已多次直接或间接地在使用它，比如键盘控制的简单应用中（参见 3.1.2 节），我们调用了 bioskey()库函数，它实际上调用了 BIOS 中断 INT16H 的功能，同样我们也可通过 int86()直接调用 INT16H。又如 3.2.2 节鼠标的使用等。总之，我们还是调用了系统的中断服务程序。在大型程序的实现中我们往往需要自己编写中断。在中断向量表中，有些中断向量，用户是可以用在自己的程序中的，如 60~67H 号中断。绝对地址为 180~19FH 的一段，是专为用户保留的，它可用作用户的软中断。如想使用中断号为 60H 的软中断，用户只要在 4×60H=180H 和 181H 处填入中断服务程序的偏移地址， 在 182H~183H 单元填入段地址即可。这样在程序中执行到 geninterrupt(0x60)时，便执行这个中断服务程序。对于硬件中断，若要通过 IRQi 线产生硬中断，用户必须制作接口电路，以能产生由低到高的中断请求信号，并通过插头接到微机的扩充槽相应的 IRQ5 脚上，用户可用的是 IRQ2、IRQ10、IRQll、IRQl2 和 IRQl5，它们的中断向量号分别是 10、72、73、74 和 77，因此可以在中断向量表中填入硬中断服务程序的段和偏移值。上述的硬件中断向量号若不用作硬件中断时，也可将其当作软件中断，不过产生中断要靠发软件中断命令，此时相应的 IRQi 脚就无作用了。



这部分我们将介绍这一更高级的中断技术，即如何用 Turbo C 实现自己的中断服务。

## 4.1 编写自己的中断程序

用 Turbo C 实现编写中断程序的方法可用三部分来实现：即编写中断服务程序、安装中断服务程序、激活中断服务程序，下面分述之。

### 4.1.1 编写中断服务程序

我们的任务是，当产生中断后，脱离被中断的程序，使系统执行中断服务的程序，它必须打断当前执行的程序，急需完成一些特定操作，因此该程序中应包括一些能完成这些操作的语句和函数。

由于产生中断时，必须保留被中断程序中断时的一些现场数据，即保存断点，这些值都在寄存器中(若不保存，当中断服务程序用到这些寄存器时，将改变它的值)，以便恢复中断时，使这些值复原，以继续执行原来中断了的程序。

Turbo C 为此提供了一种新的函数类型 interrupt，它将保存由该类型函数参数指出的各寄存器的值，而在退出该函数，即中断恢复时，再复原这些寄存器的值，因而用户的中断服务程序必须定义成这种类型的函数。如中断服务程序名定为 myp，则必须将这个函数说明成这样：

```
void interrupt myp(unsigned bp，unsiened di，unsigned si，unsigned ds，unsigned es,
unsigned dx，unsigned cx，unsigned bx，unsigned ax，unsigned ip,
unsigned cs，un3igned flags);
```

若是在小模式下的程序，只有一个段，在中断服务程序中用户就可以像用无符号整数变量一样，使用这些寄存器。

若中断服务程序中不使用上述的寄存器，也就不会改变这些寄存器原来的值，因而也就不需保存它们，这样在定义这种中断类型的函数时，可不写这些寄存器参数，如可写成：

```
void interrupt myp()
{
```

…
}
对于硬中断，则在中断服务程序结束前要送中断结束命令字给系统的中断控制寄存器，
其口地址为 0x20，中断结束命令字也为 0x20，即
outportb(0x20，0x20)；
在中断服务程序中，若不允许别的优先级较高的中断打断它，则要禁止中断，可用函
数 disable()来关闭中断。若允许中断，则可用开中断函数 enable()来开放中断。
77
4.1.2 安装中断服务程序
定义了中断服务函数后，还需将这个函数的入口地址填入中断向量表中，以便产生中
断时程序能转入中断服务程序去执行。为了防止正在改写中断向量表时，又产生别的中断而
导致程序混乱，可以关闭中断，当改写完毕后，再开放中断。一般的，常定义一个安装函数
来实现这些操作，如：
void install(void interrupt (*faddr)()，int inum)
{
disable()；
setvect(inum，faddr);
enable();
}
其中 faddr 是中断服务程序的入口地址，其函数名就代表了入口地址，而 inum 表示中
断类型号。setvect()函数就是设置中断向量的函数，上述定义的 install()函数，将完成把中断
服务程序入口地址填入中断向量 inum 中去。
setvect(intnum，faddr)会把第 intnum 号中断向量指向所指的函数，也即指向 faddr()这个
函数，faddr()必须是一个 interrupt 类型的函数。
getvect(intnum)会返回第 int num 号中断向量的值，此即 intnum 号中断服务程序的进入
地址(4 Bytes 的 far 地址)。该地址是以 pointer to function 的形式返回的。getvect()使用的方式
为：
ivect =getvect(intnum);
其中 ivect 是一个 fuction 的 pointer，存放 function 的地址，其类型必须是 interrupt 类型，
由于声明时，此 pointer 无固定值，所以冠以 void。
4.1.3 中断服务程序的激活
当中断服务程序安装完后，如何产生中断，从而执行这个中断服务程序呢？对硬件中
断，就要在相应的中断请求线(IRQi，i=0，l，2，?，7)产生一个由低到高的中断请求电平，
这个过程必须由接口电路来实现，但如何激励它产生这个电平呢？这可以用程序来控制实
现，如发命令(outportb(口地址,命令))。然后主程序等待中断，当中断产生时，便去执行中断。
对于软中断，有几种调用方法。由于中断类型的函数不同于用户定义的一般函数，因
此也不能用调用一般函数的方法来调用它，一般软中断调用可用如下方法：
1）使用库函数 geninterrupt(中断类型号)
在主函数中适当的地方，用 setvect 函数将中断服务程序的地址写入中断向量表中，然
后在需要调用的地方用 geninterrupt()函数调用。
2）直接调用
如已用 setvect(类型号，myp)设置了中断向量值，则可用 myp()；直接调用，或用指向
地址的方法调用：
(* myp)();
3）也可用在 Turbo C 程序中插入汇编语句的方法来调用，如：
setvect (inum，myp)
...
asm int inum；

...

不过用这种方法的程序生成执行程序稍麻烦点。

通常上述的调用可定义成一个中断激活函数来完成，该函数中可附加一些别的操作，

78

主程序在适当的地方调用它就可以了。

4）恢复被修改的中断向量

这一步视情况而定，当用户采用系统已定义过的中断向量，并且将其中断服务程序进行了改写，或用新的中断服务程序代替了原来的中断服务程序，为了在主程序结束后，恢复原来的中断向量以指向原中断服务程序，可以在主程序开始时，存下原中断向量的内容，这可以用取中断向量函数 getvect()来实现，如 j=(char *)getvect(0xlc)，这样 j 指针变量中将是 0xlc 中断服务程序的入口地址，由于 DOS 已定义了 0xlc 中断的服务程序入口地址，但它是一条无作用的中断服务，因而我们可以利用 0xlc 中断来完成一些用户想执行的一些操作，实际上就是用户自己的中断服务程序代替了原来的。当主程序要结束时，为了保持系统的完整性，我们可以恢复原来的中断服务入口地址，如可用

setvect(0xlc，j)

也可以再调用 install()函数再一次进行安装。一般情况下可以不加这一步。

PC286、386、486 上使用的硬中断请求信号和对应的中断向量号与外接的设备如表 3-17 所示。

表 3-17 PC X86 硬中断搞求信号与中断向量号及外设的关系

中断请求信号 中断向量号 使用的设备

IRQ0 8 系统定时器

IRQ1 9 键盘

IRQ2 10 对 XT 机保留，AT 总线扩充为 IRQ8-15

IRQ3 11 RS—232C(COM2)

IRQ4 12 RS-232C(COMl)

IRQ5 13 硬盘中断

IRQ6 14 软盘中断

IRQ7 15 打印机中断

IRQ8 70 实时钟中断

IRQ9 71 软中断方式重新指向 IRQ2

IRQ10 72 保留

IRQ11 73 保留

IRQ12 74 保留

IRQ13 75 协处理器中断

IRQ14 76 硬盘控制器

IRQ15 77 保留

中断优先级从高到低排列顺序为：IRQ0，IRQl，IBQ 2，IRQ8，IRQ9，IRQ10，IRQll，IRQl2，IRQl3，IRQl4，IRQl5，IRQ3，IRQ4，IRQ5，IRQ6，IRQ7。

4.2 问题实现

编写新的键盘中断程序要做以下几项工作：

1) 从键盘 I/O 端口 60H 读取一个字节的扫描码，并将它存入一个全局变量中供 main 程序处理，或者将它存入一个数据表中。

2) 读取控制寄存器 61H，并用 80H 完成一个 OR 操作。

3) 将结果写回控制寄存器端口 61H。

4) 在控制寄存器上用 7fh 完成一个 AND 操作，以便复位键盘触发器，告诉硬件一个按键已被处理，可以读下一个键了。

79

5) 复位中断控制器，向端口 20h 写一个 20h。

我们先定义一组宏常量记录键值，它包括 128 个键盘扫描码，然后定义两个字符型数组来保存键盘状态：

```
char key_state[128], key_pressed[128];
```

其中 key_state[128]用来表示键的当前状态，key_pressed[128]里保存的值表示哪些键被按下，值 1 表示按下， 0 表示放开。

在挂上新的键盘中断以前，将原来的键盘中断程序地址保存好，以便在程序运行结束后恢复它，我们定义一个中断指针来存放原来的地址：

```
void interrupt far (*OldInt9Handler)();
```

1）新的键盘中断程序：

```
void far interrupt NewInt9(void)
{
unsigned char ScanCode,temp;
ScanCode=inportb(0x60);
temp=inportb(0x61);
outportb(0x61,temp | 0x80);
outportb(0x61,temp & 0x7f);
if(ScanCode&0x80)
{
ScanCode&=0x7f;
key_state[ScanCode]=0;
}
else
{
key_state[ScanCode]=1;
key_pressed[ScanCode]=1;
}
outportb(0x20,0x20);
}
```

2）安装新的键盘中断程序的函数：

```
void InstallKeyboard(void)
{
int i;
for(i=0;i<128;i++)
key_state[i]=key_pressed[i]=0;
OldInt9Handler=getvect(9);
setvect(9,NewInt9);
}
```

3）恢复旧的键盘中断程序的函数：

80

```
void ShutDownKeyboard(void)
{
setvect(9,OldInt9Handler);
}
```

4）读取按键状态的函数（游戏中调用来确定按了哪些键）：

```
int GetKey(int ScanCode)
{
int res;
res=key_state[ScanCode]|key_pressed[ScanCode];
```

key_pressed[ScanCode]=0;

return res;

}

这样的话我们的程序改成这样：

while(非程序退出条件)

{

for(i = 0; i < 一定时间内; ++i)

{

if(GetKey（KEY_UP）) 处理 1；

if(GetKey(KEY_S)) 处理 2；

…

}

自动掉下。

}

那么即使在激战中一方死按着向上键，另外一方也能通过按 S 键使方块下落。而且两个按键都是有效的，即使两个人都按住不放也都能响应，具体源程序清单（例程 3-34）如下：

/*例程 3-34*/

#define KEY_A 0x1E

#define KEY_B 0x30

#define KEY_C 0x2e

#define KEY_D 0x20

#define KEY_E 0x12

#define KEY_F 0x21

#define KEY_G 0x22

#define KEY_H 0x23

#define KEY_I 0x17

#define KEY_J 0x24

#define KEY_K 0x25

#define KEY_L 0x26

#define KEY_M 0x32

#define KEY_N 0x31

#define KEY_O 0x18

#define KEY_P 0x19

81

#define KEY_Q 0x10

#define KEY_R 0x13

#define KEY_S 0x1f

#define KEY_T 0x14

#define KEY_U 0x16

#define KEY_V 0x2f

#define KEY_W 0x11

#define KEY_X 0x2d

#define KEY_Y 0x15

#define KEY_Z 0x2c

#define KEY_1 0x02

#define KEY_2 0x03

#define KEY_3 0x04

```
#define KEY_4 0x05
#define KEY_5 0x06
#define KEY_6 0x07
#define KEY_7 0x08
#define KEY_8 0x09
#define KEY_9 0x0a
#define KEY_0 0x0b
#define KEY_DASH 0x0c /* _- */
#define KEY_EQUAL 0x0d /* += */
#define KEY_LBRACKET 0x1a /* {[ */
#define KEY_RBRACKET 0x1b /* }] */
#define KEY_SEMICOLON 0x27 /* :; */
#define KEY_RQUOTE 0x28 /* "' */
#define KEY_LQUOTE 0x29 /* ~` */
#define KEY_PERIOD 0x33 /* >. */
#define KEY_COMMA 0x34 /* <, */
#define KEY_SLASH 0x35 /* ?/ */
#define KEY_BACKSLASH 0x2b /* |\ */
#define KEY_F1 0x3b
#define KEY_F2 0x3c
#define KEY_F3 0x3d
#define KEY_F4 0x3e
#define KEY_F5 0x3f
#define KEY_F6 0x40
#define KEY_F7 0x41
#define KEY_F8 0x42
#define KEY_F9 0x43
#define KEY_F10 0x44
#define KEY_ESC 0x01
#define KEY_BACKSPACE 0x0e
#define KEY_TAB 0x0f
```

```
#define KEY_ENTER 0x1c
#define KEY_CONTROL 0x1d
#define KEY_LSHIFT 0x2a
#define KEY_RSHIFT 0x36
#define KEY_PRTSC 0x37
#define KEY_ALT 0x38
#define KEY_SPACE 0x39
#define KEY_CAPSLOCK 0x3a
#define KEY_NUMLOCK 0x45
#define KEY_SCROLLLOCK 0x46
#define KEY_HOME 0x47
#define KEY_UP 0x48
#define KEY_PGUP 0x49
#define KEY_MINUS 0x4a
#define KEY_LEFT 0x4b
#define KEY_CENTER 0x4c
```

```
#define KEY_RIGHT 0x4d
#define KEY_PLUS 0x4e
#define KEY_END 0x4f
#define KEY_DOWN 0x50
#define KEY_PGDOWN 0x51
#define KEY_INS 0x52
#define KEY_DEL 0x53
char key_state[128],key_pressed[128];
void interrupt far (*OldInt9Handler)();
void far interrupt NewInt9(void)
{
unsigned char ScanCode,temp;
ScanCode=inportb(0x60);
temp=inportb(0x61);
outportb(0x61,temp | 0x80);
outportb(0x61,temp & 0x7f);
if(ScanCode&0x80)
{
ScanCode&=0x7f;
key_state[ScanCode]=0;
}
else
{
key_state[ScanCode]=1;
key_pressed[ScanCode]=1;
}
83
outportb(0x20,0x20);
}
void InstallKeyboard(void)
{
int i;
for(i=0;i<128;i++)
key_state[i]=key_pressed[i]=0;
OldInt9Handler=getvect(9);
setvect(9,NewInt9);
}
void ShutDownKeyboard(void)
{
setvect(9,OldInt9Handler);
}
int GetKey(int ScanCode)
{
int res;
res=key_state[ScanCode]|key_pressed[ScanCode];
key_pressed[ScanCode]=0;
return res;
}
```

```
void main()
{
int i = 0;
int j = 0;
int k = 0;
int m = 0;
InstallKeyboard();
for(i = 0; i < 10000; ++i)
for(j = 0; j < 10000; ++j)
{
for(k = 0; k <10000; ++k)
for(m = 0; m <10000; ++m);
if(GetKey(KEY_UP))
{
printf("pressed key UP\n");
}
if(GetKey(KEY_S))
{
84
printf("pressed key S\n");
}
}
ShutDownKeyboard();
}
```

4.3 其它应用——硬中断演示秒表程序

这个程序(例程 3_35)是一个硬中断程序，我们知道 PC 系统以每秒 18.2 次的频率进行时钟硬中断(使用中断请求 IRQ8)，即执行 8 号中断。这个中断周而复始的在进行，在它的中断服务程序中除了进行日时钟计数和磁盘驱动器超时检测控制外，接着又进行 0xlc 的软中断调用，0xlc 软中断只有一条返回指令，它不作什么事情，因而我们可以改写它的内容，使其变为一个有用的软中断服务程序。在这个例子中，我们利用每秒 18.2 次的定时硬中断每秒要调用 18.2 次的软中断 0xlc，将中断 0xlc 中断服务程序改写为对进入该中断的次数进行计数的程序，每到 18 次时，在屏幕的右上角开一个窗口(window(50，1，54，3))，在窗口的中间位置显示 0~9 十个数字中的一个，频率接近于秒表数(不过只显示十个数)。由于这是一个硬中断演示程序，计时并不准确，若要精确计时，则应 91 次 0xlc 中断为 5 秒。

该程序中用 programsize 表示程序长度，设置为 400 节，由于在小模式下，Turbo C 使用 64K 的一个段。

```
/*例程 3_35*/
#include <dos.h>
#include <conio.h>
#define programsize 400
#define TRUE 1
void interrupt(*oldtimer)();
void interrupt newtimer();
void install();
static struct SREGS seg;
int b=0;int b1=0;
unsigned intsp,intss;
unsigned myss,stack,x0,y0;
```

```
int busy=0;
void on_timer();
void goxy();
void rexy();
void prt();
main()
{
char ch;
char *p;
p=(char *)newtimer;
on_timer(p);
while(TRUE)
{
ch=getch();
switch(ch){ /* 键盘输入 0、1 或 q 进行功能选择 */
85
case '0':
install(oldtimer);
break;
case '1':
b1=47;
install(newtimer);
break;
case 'q':
install(oldtimer);
exit(1);
default:
printf("%c",ch); /* 若是其它字符就打印出来 */
}
}
}
void on_timer(int (*p)()) /* p 指向中断函数 */
{
segread(&seg);
stack=programsize*16; /* 设置堆栈 */
myss=_SS;
oldtimer=getvect(0x1c); /* 得到原 0xlc 中断向量 */
}
void install(void interrupt (* faddr) () )
{
disable();
setvect(0x1c,faddr); /* 设置 0xlc 新中断向量 */
enable();
}
void interrupt newtimer()
{
(* oldtimer)(); /* 指向旧中断程序 */
if(busy==0) {
```

```
busy=1;
disable();
intsp=_SP;
intss=_SS; /* 保存 SP,SS 值 */
_SP=stack;
_SS=myss; /* 设置 SP,SS 为新值 */
enable();
rexy(); /* 得到当前光标位置 */
clrscr();
window(50,1,54,3);
86
textcolor(YELLOW);
textbackground(RED);
b+=1; /* 如果中断 18 次,则打印秒计数值 */
if(b==18) {
gotoxy(3,2);
b1++;
b=0;
if(b1>57)b1=48; /* 计数值超过 9,则令 bl 为 0 */
prt(b1); /* 显示计数值 */
}
goxy(x0,y0); /* 光标返回原处 */
disable();
_SP=intsp; /* 恢复 SP,SS 的原值 */
_SS=intss;
enable();
busy=0;
}
}
void goxy(int x,int y) /* 光标回到 x,y 处 */
{
union REGS rg;
rg.h.ah=2;
rg.h.dl=y;
rg.h.dh=x;
rg.h.bh=0;
int86(0x10,&rg,&rg);
}
void rexy() /* 得到光标坐标 */
{
union REGS rg;
rg.h.ah=3;
rg.h.bh=0;
int86(0x10,&rg,&rg);
y0=rg.h.dl;
x0=rg.h.dh;
}
void prt(p) /* 显示字符 */
```

```
{
union REGS rg;
rg.h.al=p;
rg.h.ah=14;
int86(0x10,&rg,&rg);
87
}
```

实验四

(1) 完善扫雷游戏，考虑游戏时间的数据显示。

(2) 完善扫雷游戏，完成 Windows 程序中扫雷英雄榜的功能。

(3) 考虑是否将挖雷游戏扩展为对抗版。

5 发声技术

[问题的提出] 乐谱的 1、2、3、4、5、6、7，加上高低音可以谱出动听的曲子，请编写程序，使计算机能够播放歌曲。

[分析] 播放歌曲意味着让计算机发声，声音从 PC 机内的扬声器发出，所以这个问题将与硬件扬声器电路有关。

[解答] 解决这一编程问题，让我们首先简单了解一下计算机发声的原理。在 PC 机的系统板上装有定时与计数器 8253 芯片，还有 8255 可编程并行接口芯片，由它们组成的硬件电路可用来产生 PC 机内扬声器的声音，对于 286、386、486、586 等 PC 微机，由于采用了超大规模集成电路，因而看不到这些芯片，它们均集成在外围电路芯片上了。

当我们操作计算机时，常常听到的发声，就是由软件控制这些电路而产生的。声音的长短和音调的高低，均可由程序进行控制。在扬声器电路中，定时器的频率决定了扬声器发音的频率，所以可通过设定定时器电路的频率来使扬声器发出不同的声音。对定时器电路进行频率设定时，首先对其命令寄存器(口地址为 0x43)写命令字，如写入 0xb6，这可用 outporb(0x43，0xb6)；来实现，则表示选择该定时器的第二个通道，计数频率先送低 8 位(二进制)，后送高 8 位。接着用口地址 0x42 送频率计数值，先送低 8 位，后送高 8 位，即用 outportb(0x42，低 8 位频率计数值)和 outportb(0x42，高 8 位频率计数值)来实现。通过这两步使定时器电路产生一系列方波信号，此信号是否能推动扬声器发音，还要看由 8255 产生的门控信号和送数信号是否为 1，而它们也可编程，口地址为 0x61。为了不影响 8255 口地址 61H 中的其他高位，应先输入口地址 6lH 的现有值 bits，即用 bits= inportb(0x61)来实现，然后就可用 outportb(0x61，bits|3)来允许发声，而用 outportb(0x61，bits&0xfc)来禁止发声，且不改变 8255 其它位原来的值，关于这方面的详细内容可以参阅 IBM PC/XT 接口技术方面书籍有关内容。

5.1 声音函数

编写音乐程序播放歌曲，最简单的方法是可以直接使用 TURBO C 在 dos.h 中提供的有关发声的函数 sound()和 nosound()。sound()函数用于产生声音，其原型如下：

void sound(unsigned frequency)；

该函数的入口参数为扬声器要产生声音的频率。

与 sound()函数相反，nosound ()函数用于关闭扬声器，其原型为：

void nosound(void)；

该函数没有入口和出口参数，它只是简单地把口地址 61H 中的低 2 位清 0。

在利用函数 sound 产生指定频率的声音后，一般要过一段时间后再调用函数 nosound 关闭扬声器，这样我们才能清楚地听到一个声音。如果扬声器刚打开就关闭，我们是很难听到一个声音的。某个频率的声音延续时间的长短是重要的，它将直接影响音响效果。这需要使用 TURBOC 提供了专门的延时函数 delay，其原型说明如下：

void delay (unsigned milliseconds)；

该函数中断程序的执行，中断的时间由 milliseconds 指定。

例程 3-36：该程序每间隔 10000 milliseconds pc 扬声器发出不同频率的声音，直到频率大于 5000hz。

```
#include<dos.h>
main()
{
int freq;
for(freq=50;freq<5000;freq+=50)
{
sound (freq);
delay(10000);
}
nosound();
}
```

如果不能使用上述现成的函数 sound()和 nosound()，当然我们也可以采用上节中的方法，用 I/O 接口的输入输出函数，自己编写产生声音和关闭声音的函数。下面可供参考的函数 SOUND()与 TURBOC 提供的产生声音函数 sound()的算法类似：首先函数 SOUND()中使用了一个由一个整数和两个字符组成的联合，其目的在于方便地把一个 16 位数分解成两个 8 位数。为了打开扬声器，需要把口地址 61H 的低 2 位置位，但又不能影响其他高位，为此，先输入口地址 61H 中的现有值，与 3 逻辑或后再输出到口地址 61H。

```
void SOUND(unsigned frequency)
{
union { /* 定义由一个整数和两个字符组成的联合 */
unsigned divisor;
unsigned char c[2];
} tone;
tone.divisor=119328/frequency； /* 计算该频率对应的定时器计数值 */
outportb(0x43，0xb6)； /* 通知定时器采用新的计数 */
outportb(0x42，tone.c[0])； /* 计数低字节先送到定时器 */
outportb(0x42，tone.c[1]); /* 计数高字节后送到定时器 */
outportb(0x61, inportb(0x61) | 3 ); /* 使定时器到喇叭的输出有效 */
}
```

如下供参考的函数 NOSOUND()，为了不影响口地址 61H 中的其他高位，应先输入口地址 6lH 的现有值．在屏蔽掉低 2 位后再输出到口地址 61H。

```
void NOSOUND(void)
{
outportb(0x61，inportb(0x61) & 0xfc)); /* 使定时器到喇叭的输出无效 */
}
```

5.2 计算机乐谱

表 3-18 是频率与音阶的对照表。我们可以通过该表编制出自己的驱动程序。编制乐谱程序一般在原乐谱的基础上添加一些控制字符来完成。如：

。。。。。。。。

2 6 2 23 3 6 7 1 11

就是孟庭苇演唱的“羞答答的玫瑰静俏俏地开”的第一句歌词的乐谱。

在计算机中可以表述为：

89

600 H1 0.5 H1 0.5 H1 0.5 M7 0.5 M6 1 H3 1 H2 0.5 H3 0.5 H2 0.5 M6 0.5 H2 2

第一个为音长基准，一般为 300，600，900，1200。后面字 H1 表示高音的 1，音阶的设置如下：

最高音：在音符前加“E”；

高音：在音符前加“计”；

中音：在音符前加“M”；

低音：在音符前加“L”；

再后面的字为节拍数，其中的 0.5 表示节拍数，每个音的音长＝音长基数×节拍数，如第一个“1”的音长为 600×0.5＝300。

表 3-18 频率与音阶的对照表

| 低音 | 频率 | 中音 | 频率 | 高音 | 频率 | 最高音 | 频率 |
|---|---|---|---|---|---|---|---|
| 1 | 131 | 1 | 262 | 1 | 523 | 1 | 1047 |
| 2 | 147 | 2 | 296 | 2 | 587 | 2 | 1175 |
| 3 | 165 | 3 | 330 | 3 | 659 | 3 | 1319 |
| 4 | 176 | 4 | 349 | 4 | 699 | 4 | 1397 |
| 5 | 196 | 5 | 392 | 5 | 784 | 5 | 1568 |
| 6 | 220 | 6 | 440 | 6 | 880 | 6 | 1760 |
| 7 | 247 | 7 | 494 | 7 | 988 | 7 | 1976 |

知道了这些知识，就容易编制一个乐谱驱动程序。思路是将各个频率存储在一个二维数组中，根据音阶字符、音符和节拍数，得到发音的音长，使用 sound 函数发音，使用 delay 函数控制。

5.3 问题实现

5.3.1 调用 sound()和 unsound()

下面的程序（例程 3-37）先开辟两个数组 freq[96]和 dely[96]分别用于存储声音的频率和延时。采用图形方式，利用 printtext()函数在屏幕上分别显示字符串 Welcome !"，"Please press any key to start ......"，"Enjoy yourself !”，"Press any key to end !" 和 "Thank you! Bye Bye ....."

printtext()定义如下：

```
void printtext(unsigned char *temp[],int i)
{
setcolor(4); //设置颜色为 red
settextstyle(TRIPLEX_FONT,HORIZ_DIR,3); // 设置字符的字体，方向和大小
outtextxy(100,40+i*50,temp[i]); // 显示字符串
}
```

另外，在此程序中还调用了 conio.h 中的函数 kbhit(),用于判断是否有键按下，当没有键按下时返回 0。

```
/*例程 3-37*/
#include<dos.h>
#include<graphics.h>
void printtext(unsigned char *temp[],int i); // 用于在屏幕上显示字符串
main()
{
90
int i,graphdriver,graphmode;
unsigned char *temp[4];
int freq[96]={784,660,588,660,523,523,
588,494,440,523,392,392,
330,392,440,523,784,440,523,392,
784,1048,880,784,660,784,588,588,
588,660,494,440,392,440,523,588,
330,523,440,392,440,523,392,392,
```

```
660,784,494,588,440,523,392,392,
330,392,330,392,392,440,494,588,440,440,392,440,
523,588,784,660,588,660,588,523,440,392,
330,523,440,523,440,392,330,392,440,523,
392,392,660,784,588,660,588,523,494,440,784,784};
int dely[96]={25,50,12,12,50,50,
25,50,12,12,50,50,
50,38,12,38,12,12,12,25,
38,12,12,12,12,12,50,50,
38,12,25,25,38,12,25,25,
25,25,12,12,12,12,50,50,
38,12,25,25,12,12,50,25,
12,12,12,12,12,12,12,12,50,25,12,12,
38,12,25,25,25,12,12,25,12,12,
50,50,12,12,12,12,12,12,12,12,
50,25,12,12,12,12,12,12,25,25,50,50};
graphdriver=DETECT;
graphmode=0;
temp[0]="Welcome !";
temp[1]="Please press any key to start ......";
temp[2]="Enjoy yourself !";
temp[3]="Press any key to end !";
temp[4]="Thank you! Bye Bye ......";
initgraph(&graphdriver,&graphmode,""); //系统初始化
cleardevice(); //清除屏幕
settextjustify(LEFT_TEXT,CENTER_TEXT); //设置字符排列方式
for(i=0;i<2;i++)
printtext(temp,i);
getch();
for(i=2;i<4;i++)
printtext(temp,i);
i=0;
while(i<96&&!kbhit())
91
{
sound(freq[i]); // 扬声器根据频率发声
delay(1100*dely[i]); // 声音延时
i++;
}
nosound(); // 关闭扬声器
printtext(temp,4);
getch();
closegraph(); // 关闭图形模式
}
void printtext(unsigned char *temp[],int i)
{
setcolor(4); // 设置颜色
settextstyle(TRIPLEX_FONT,HORIZ_DIR,3); // 设置字符的字体，方向和大小
```

```
outtextxy(100,40+i*50,temp[i]); // 在所指定的坐标出显示字符串
}
```

5.3.2 调用 inportb()和 outportb()

这个程序（例程 3-38）利用 3.5.2 节的 SOUND()和 UNSOUND()改写前一实现,播放一段不同的曲目。

```
/*例程 3-38*/
#include<dos.h>
#include<graphics.h>
void printtext(unsigned char *temp[],int i);
void SOUND(unsigned frequency);
void NOSOUND(void);
main()
{
int i,graphdriver,graphmode;
unsigned char *temp[4];
int freq[87]={196,262,262,262,330,294,262,294,330,294,262,
330,394,440,440,394,330,330,262,294,262,294,
330,294,262,230,230,196,262,440,394,330,330,
262,294,262,294,440,394,330,330,394,440,523,
394,330,330,262,294,262,294,330,294,262,230,
230,196,262,440,394,330,330,262,294,262,294,
440,394,330,330,394,440,523,394,330,330,262,
294,262,294,330,294,262,230,230,196,262};
int dely[87]={25,38,12,25,25,38,12,25,12,12,56,25,25,50,25,
38,12,12,12,38,12,25,12,12,38,12,25,25,100,25,
38,12,12,12,38,12,25,25,38,12,25,25,100,25,38,
12,12,12,38,12,25,12,12,38,12,25,25,100,25,38,
92
12,12,12,38,12,25,25,38,12,25,25,100,25,38,12,
12,12,38,12,25,12,12,38,12,25,25,100
};
graphdriver=DETECT;
graphmode=0;
temp[0]="Welcome !";
temp[1]="Please press any key to start ......";
temp[2]="Enjoy yourself !";
temp[3]="Press any key to end !";
temp[4]="Thank you! Bye Bye ......";
initgraph(&graphdriver,&graphmode,"");
cleardevice();
settextjustify(LEFT_TEXT,CENTER_TEXT);
for(i=0;i<2;i++)
printtext(temp,i);
getch();
for(i=2;i<4;i++)
printtext(temp,i);
i=0;
while(i<87&&!kbhit())
```

```
{
SOUND(freq[i]);
delay(1100*dely[i]);
i++;
}
NOSOUND();
printtext(temp,4);
getch();
closegraph();
}
void printtext(unsigned char *temp[],int i)
{
setcolor(4);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
outtextxy(100,40+i*50,temp[i]);
}
void SOUND(unsigned frequency)
{
union { /* 定义由-个整数和两个字符组成的联合 */
93
unsigned divisor;
unsigned char c[2];
} tone;
tone.divisor=119328L/frequency; /* 计算该频率对应的定时器计数值 */
outportb(0x43,0xb6); /* 通知定时器采用新的计数 */
outportb(0x42,tone.c[0]); /* 计数低字节先送到定时器 */
outportb(0x42,tone.c[1]); /* 计数高字节后送到定时器 */
outportb(0x61,inportb(0x61) | 3 ); /* 使定时器到喇叭的输出有效 */
}
void NOSOUND(void)
{
outportb(0x61,(inportb(0x61) & 0xfc)); /* 使定时器到喇叭的输出无效 */
}
```

实验五

(1) 完善扫雷游戏，当一轮游戏成功时，播放一段优美的音乐。

(2) 完善扫雷游戏，当触雷时能发出爆炸或其它适合情景的声音。

6 汉字显示技术

[问题的提出] 到目前为止，我们编写的 C 程序，其用于人机交互的提示或菜单都是英文的，那么如何在没有汉化的 Turbo C 集成开发环境下编制显示汉字的程序呢？

[解答] 解决这一编程问题，我们首先必须了解有关汉字编码及字库的知识。根据对汉字使用频率的研究，可把汉字分成高频字(约 100 个)，常用字(约 3000 个)，次常用字(约 4000 个)，罕见字(约 8000 个)和死字(约 45000 个)，即正常使用的汉字达 15000 个。我国 1981 年公布了《通讯用汉字字符集(基本集)及其交换码标准》GB2312-80 方案，把高频字、常用字、和次常用字集合成汉字基本字符集(共 6763 个)，在该集中按汉字使用的频度，又将其分为一级汉字 3755 个(按拼音排序)、二级汉字 3008 个(按部首排序)，再加上西文字母、数字、图形符号等 700 个。

6.1 汉字编码

6.1.1 区位码

国家标准的汉字字符集(GB2312—80)在汉字操作系统中是以汉字库的形式提供的。汉字库结构作了统一规定，如图 3-19(a)所示，即将字库分成 94 个区，每个区有 94 个汉字(以位作区别)每一个汉字在汉字库中有确定的区和位编号(用两个字节)，这就是所谓的区位码(区位码的第一个字节表示区号，第二个字节表示位号，因而只要知道了区位码，就可知道该汉字在字库中的地址，每个汉字在字库中是以点阵字模形式存储的，如一般采用 16×16 点阵形式，每个点用一个二进位表示，存 1 的点，当显示时，可以在屏上显示一个亮点，存 0 的点，则在屏上不显示，这样把存某字的 16×16 点阵信息直接用来在显示器上按上述原则显示，则将出现对应的汉字，如一个"大"字的 16×16 点阵字模如图 3-19(b)所示，当用存储单元存储该字模信息时，将需 32 个字节地址，图 3-19(b)右边写出了该字模对应的字节值。"大"字区位码为 2083，表示它位于第 20 区第 84 个。

94

6.1.2 内码

汉字使用两字节表示，国家制定了统一标准，称为国标码。国标码规定，每个字节使用后面 7 位，第一位为 0。为了区别于英文的 ASCII 码，国标码在计算机上使用的时候，规定汉字每个字节第一位设置为 1，以表示该两字节为汉字，称为内码。以"大"字为例子：

国标码 3473H： 0 0 1 1 0 1 0 0 0 1 1 1 0 0 1 1

内码 B4F3H： 1 0 1 1 0 1 0 0 1 1 1 1 0 0 1 1

国标码与内码有一定的转换公式，即 16 进制的区位码，两个字节各加 80H，就成为了国标码。

汉字字模在字库中存放的位置根据汉字的区位码来确定，内码是汉字在机内的表示。由于区位码和内码存在固定的转换关系，所以当在支持汉字输入的系统中，键盘输入的汉字内码即在程序中存在，将其转换为区位码，再从字库中找到对应的汉字字模，然后再用有关的位操作和循环语句，对每个字节的每一位进行判断，如同过滤一样，如果某位是 1，则按设置的颜色在屏幕的相应位置画点（用 graphics.h 中的显示象素点的函数 putpixel()），若某位为 0，则不画点，这样就可按预先设置的颜色在相应位置显示出该汉字来。

6.1.3 内码到区位码的转换

若汉字内码为十六进制数 h2h1l2l1，则区号 qh 相位号 wh 分别为：

qh= h2h1-0xa0;

wh= l2l1-0xa0；

若用十进制表示内码为 dld2，则

qh=dl-l60；

wh=d2-160；

即区位码 qw 为：

qw=100*(d1-160)十(d2-160);

反过来，若已经知道了区位码 qw。则也可求得区号和位号：

qh=qw/100；

wh=qw-100*qh;

因而该汉字在汉字库中离起点的偏移位置(以字节为单位)，可计算为：

offset=(94*(qh-1)+(wh-1))* 32;

图 3-19 (a)国标(GB2312-80)汉字字符集 （b）"大"的字模（16×16）

95

注意：字库中每 1 区有 94 个字符。

这样，就可以找寻到文件的偏移量，读出一个 char bytes[32]数组。这样 bytes 数组中则存了要显示汉字的 16×16 点阵字模，然后将字模按行扫描的办法，通过循环用 putpixel()函数在屏幕设定位置显示出象点，因而组合成一个显示的汉字。

6.2 问题实现

下面的程序（例程 3-39）采用 3.6.1.3 介绍的转换方法，当通过 get_hz()函数取得简体汉字系统的 16*16 点阵字库（这里文件名为 HZK16，存放在作者计算机的 d:/zy/tc 下，用户可

从中文 DOS 或其它中文系统中获得中文字库）中的汉字字模后，(在 mat[]中)用 dishz()函数显示汉字时，用预先定义的一个数组 mask[]中的元素去和字模的相应位相与，若结果大于 0，则用 putpixel(x,y,color)，在 x，y 处用 color 颜色显示；若为 0，则不显示。mask[]数组中存了一个字节的每位权值，这样用 mask[j%8]&mat[pos+j/8]即得到了第[pos+j/8]字节的第[j%8]位的值，因而用该值便可控制显示。

在该程序中，用指针变量 s 作为要显示汉字串的地址指针，因而在取出汉字字模显示后，因一个汉字占两个字节，所以 s 必须加 2，以便取下面的汉字。由于在设定的图形方式下，x 的最大值可取 639，因此当该行 x 值未到 640，并且未到汉字串尾时，继续再该行显示汉字，否则令 x=20，s+=2，在下一行开始显示汉字，这是由两重 while 循环结构实现的。

```
/*例程 3-39*/
#include <stdio.h>
#include <graphics.h>
FILE *hzk_p;
void open_hzk(void);
void get_hz(char incode[],char bytes[]);
void dishz(int x,int y,char cade[],int color);
main()
{
int x=20;
int y=100;
char *s="岱宗夫如何，齐鲁青未了。造化钟神秀，阴阳割昏晓。荡胸生层云，决眦入归鸟，会当凌绝顶，一览众山小。";
int driver=DETECT;
int mode=0;
initgraph(&driver,&mode,"");
open_hzk();
while (*s!=NULL)
{
while (x<640 && (*s!=NULL))
{
dishz(x,y,s,LIGHTRED);
x+=20;
s+=2;
}
x=20;y+=20;
}
96
getch();
fclose(hzk_p);
closegraph();
}
void open_hzk()
{
hzk_p=fopen("d:/zy/tc/hzk16","rb");
if (!hzk_p){
printf("The file HZK16 not exist! Enter to system\n");
getch();
closegraph();
```

```
exit(1);
}
}
void get_hz(char incode[],char bytes[])
{
unsigned char qh,wh;
unsigned long offset;
qh=incode[0]-0xa0; /*得到区号*/
wh=incode[1]-0xa0; /*得到位号*/
offset=(94*(qh-1)+(wh-1))*32L; /*得到偏移位置*/
fseek(hzk_p,offset,SEEK_SET); /*移文件指针到要读取的汉字字模处*/
fread(bytes,32,1,hzk_p); /*读取 32 个汉字的汉字字模*/
}
void dishz(int x0,int y0,char code[],int color)
{
unsigned char mask[]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
/*屏蔽字模每行各位的数组*/
int i,j,x,y,pos;
char mat[32];
get_hz(code,mat);
y=y0;
for (i=0;i<16;++i)
{
x=x0;
pos=2*i;
for(j=0;j<16;++j)
{
/*屏蔽出汉字字模的一个位，即一个点是 1 则显示，否则不显示该点*/
if ((mask[j%8]&mat[pos+j/8])!=NULL)
97
putpixel(x,y,color);
++x;
}
++y;
}
}
```

除了用 putpixel()函数调用来显示汉字字模点阵外，也可采用 BIOS 显示象素点的功能调用来显示汉字点阵，但显示速度都较慢，还有一种显示速度最快的方法，就是直接写屏法，实际上是将要显示的汉字点阵信息在图形方式下直接存入显示存储器 VRAM 中，于是在屏上显示出汉字来。这里就不详细介绍了。

对于一些应用程序，特别是实时性强的应用程序，可以将菜单中或程序提示中的汉字放在一个专用汉字库中，而不必带有中文 DOS 系统中标准汉字库，从而可大大节省程序开销，提高运行速度。由于用到的汉字数量很少，所以建一个小型汉字库是容易的。

实验六

(1) 完善扫雷游戏，具有中文菜单和其它的中文信息显示。

(2) 练习构造自己的专用字库。